PRELIMINARY

Am386® SE

High-Performance, Low-Power, 32-Bit Embedded Microprocessor

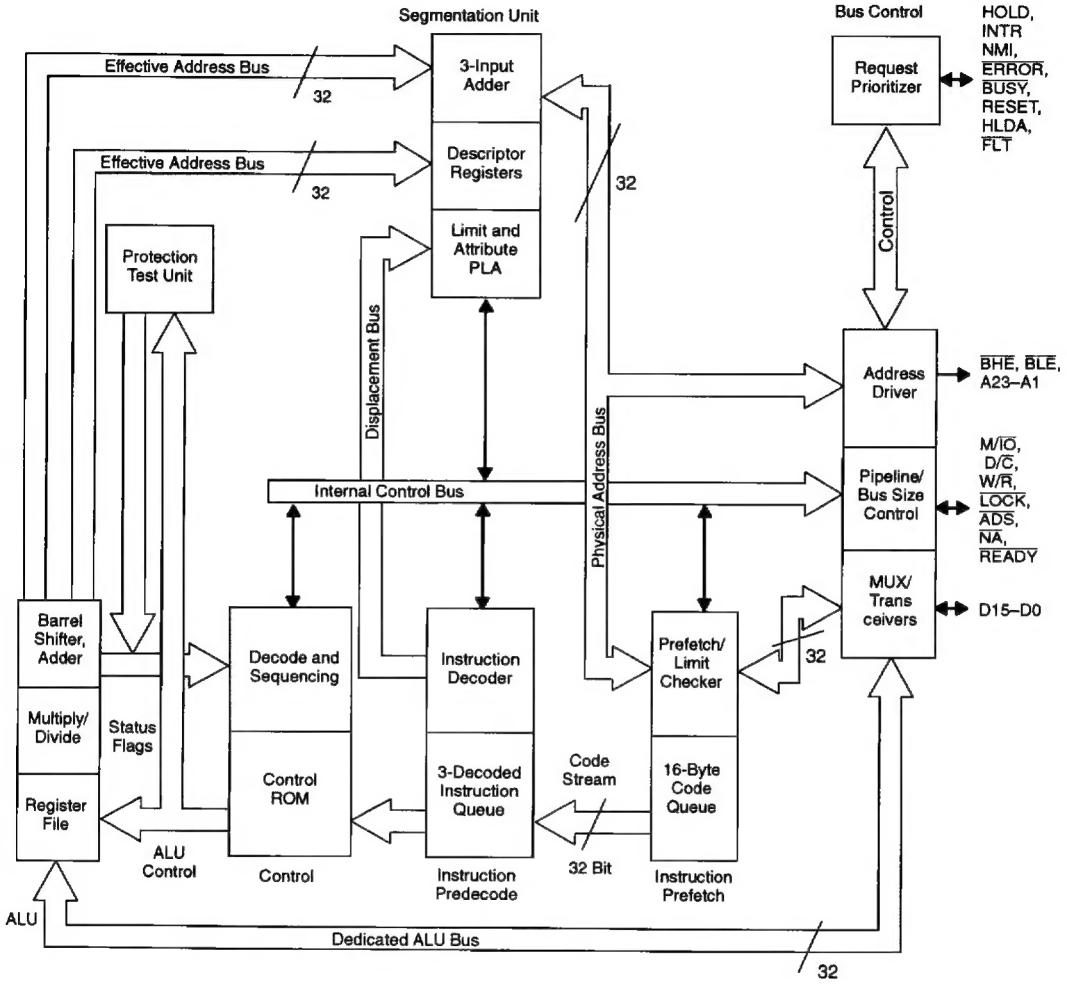
Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- **Member of Am386 E CPU Series**
  - — Am386 CPU Core
  - — Designed for embedded applications
  - — Socket compatible with 386SX processor
- **Industry standard architecture allows use of existing peripheral support chips, development tools, and application software**
- **Ideal for embedded applications**
  - — Low power consumption
  - — 3–5 V operation (25 MHz)
  - — Fully static operation
- **High performance**
  - — 25- and 33-MHz operating frequencies
  - — 32-bit internal architecture
  - — Four levels of hardware-enforced protection
- **Optimized for the cost-sensitive embedded marketplace**
  - — 16-bit data plan
  - — Real and protected mode operation without paging
  - — Full Segmentation Unit and Descriptor Table support
- **Supports world's largest software base for x86 architectures**
- **Compatible with Microsoft at Work™ and Novell NEST embedded software**
- **Coprocessor interface; supports 387SX-compatible math coprocessor**
- **Advanced packaging options**
  - — 100-pin Plastic Quad Flat Pack
  - — 100-pin Thin Quad Flat Pack
- **Extended temperature versions available**
- **Based on AMD® advanced CMOS technology**

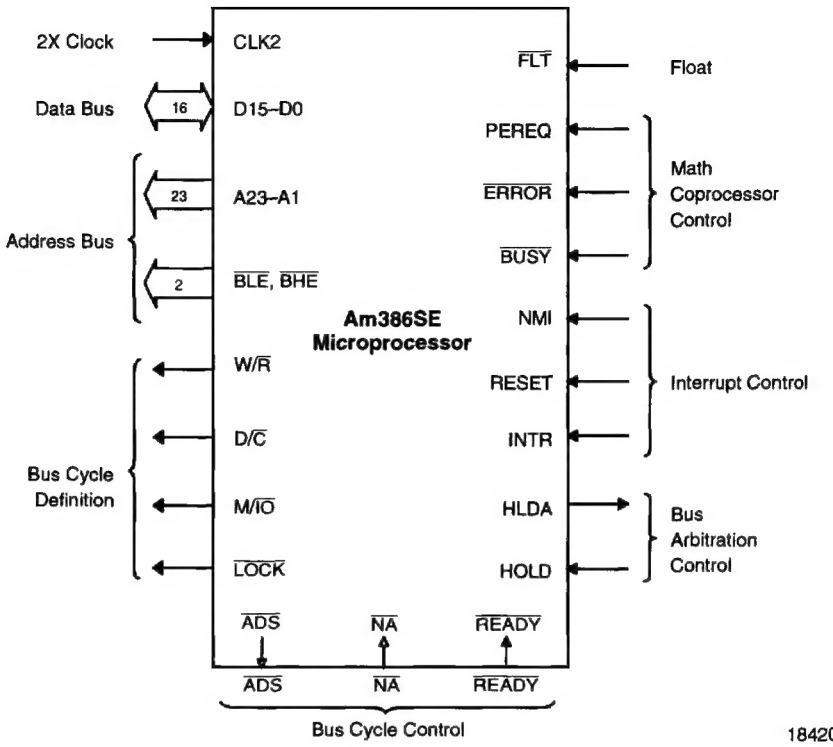
Publication#:18420 Rev. B Amendment:/0
Issue Date: June 1994

BLOCK DIAGRAM



18420A–001

LOGIC SYMBOL



FUNCTIONAL DESCRIPTION

True Static Operation

The Am386SE microprocessor incorporates a true static design. Unlike dynamic circuit design, the Am386SE CPU eliminates the minimum operating frequency restriction. It may be clocked from its maximum speed of 33 MHz all the way down to 0 MHz (DC). System designers can use this feature to design energy efficent embedded control devices.

Standby Mode

This true static design allows for a standby mode. At any operating speed (33 to 0 MHz), the Am386SE microprocessor will retain its state (i.e., the contents of all its registers). By shutting off the clock completely, the device enters standby mode. Since power consumption is proportional to clock frequency, operating power consumption is reduced as the frequency is lowered. In standby mode, typical current draw is reduced to less than 20 µA at DC.

Not only does this feature improve battery life, but it also simplifies the design in the following ways:

1. Eliminates the need for software in the BIOS to save and restore the contents of registers.
2. Allows simpler circuitry to control stopping of the clock since the system does not need to know what state the processor is in.

Lower Operating I_{CC}

True static design also allows lower operating I_{CC} when operating at any speed.

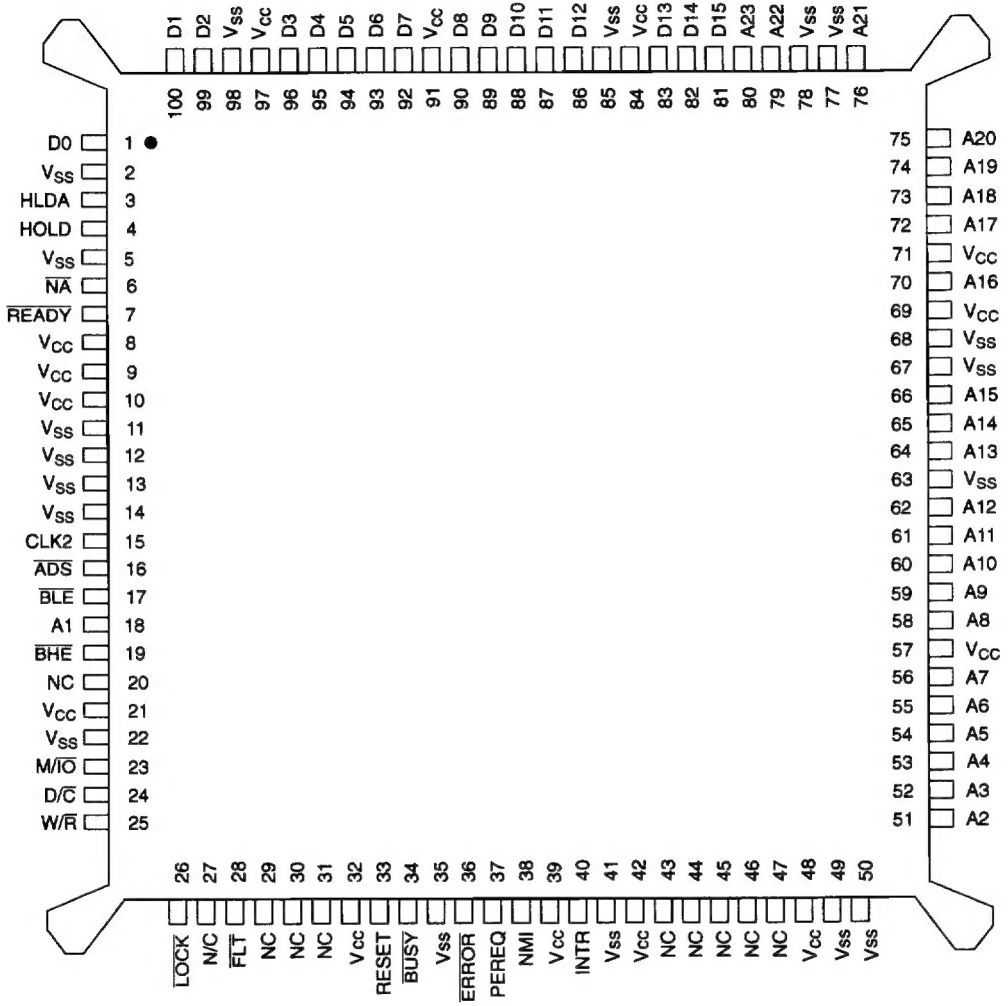
Performance On Demand (Am386SE CPU)

The Am386SE microprocessor retains its state at any speed from 0 MHz (DC) to its maximum operating speed. With this feature, system designers may vary the operating speed of the system to extend the battery life in portable systems.

For example, the system could operate at low speeds during inactivity or polling operations. However, upon interrupt, the system clock can be increased up to its maximum speed. After a user-defined time-out period, the system can be returned to a low (or 0 MHz) operating speed without losing its state. This design maximizes battery life while achieving optimal performance.

CONNECTION DIAGRAMS

Top Side View—100-Lead Plastic Quad Flat Pack and Thin Quad Flat Pack



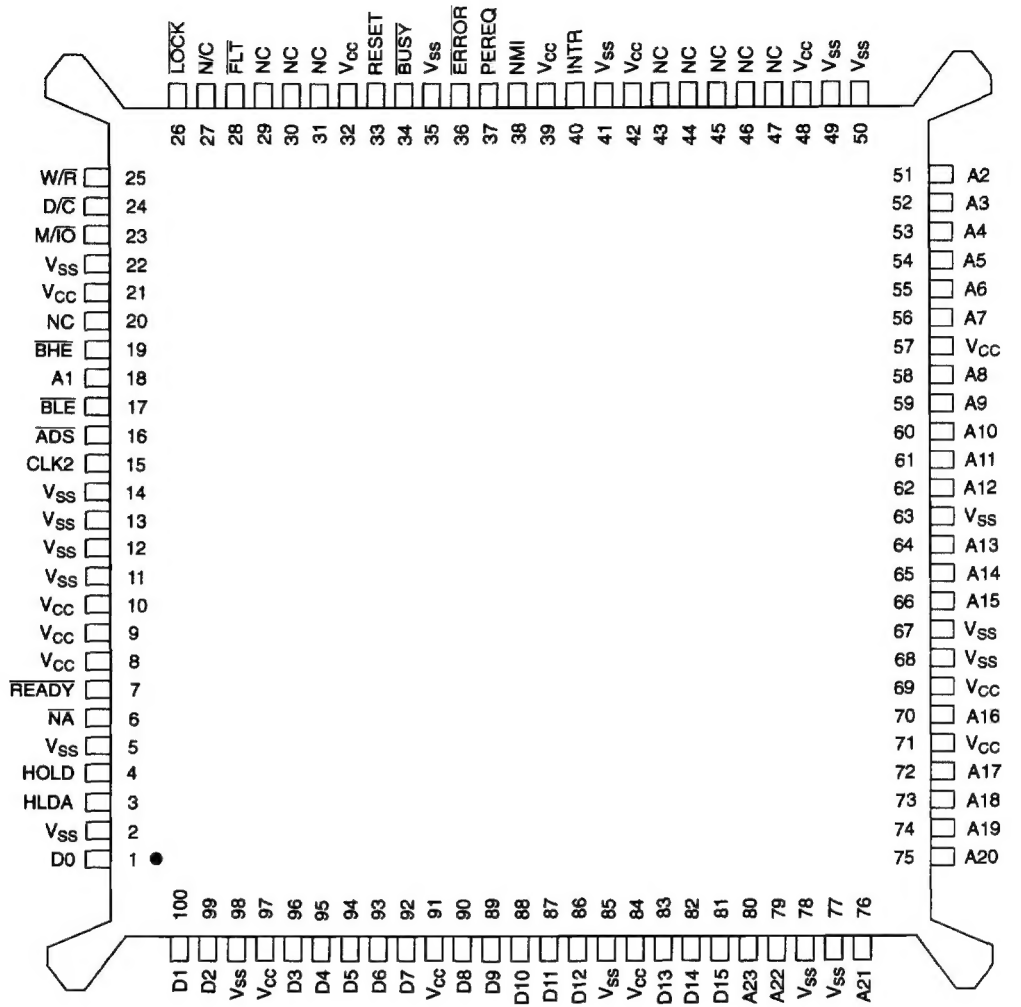
Notes:

Pin 1 is marked for orientation.

NC = Not connected; connection of an NC pin may cause a malfunction or incompatibility with future versions of the Am386SE microprocessor.

CONNECTION DIAGRAMS (continued)

Pin Side View—100-Lead Plastic Quad Flat Pack and Thin Quad Flat Pack



Notes:
Pin 1 is marked for orientation.
NC = Not connected; connection of an NC pin may cause a malfunction or incompatibility with future versions of the Am386SE microprocessor.

PIN DESIGNATION TABLES (sorted by Functional Grouping)

Address		Data		Control		NC	V_{CC}	V_{SS}
Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin No.	Pin No.	Pin No.
A1	18	D0	1	$\overline{ADS}$	16	20	8	2
A2	51	D1	100	$\overline{BHE}$	19	27	9	5
A3	52	D2	99	$\overline{BLE}$	17	29	10	11
A4	53	D3	96	$\overline{BUSY}$	34	30	21	12
A5	54	D4	95	CLK2	15	31	32	13
A6	55	D5	94	D/$\overline{C}$	24	43	39	14
A7	56	D6	93	$\overline{ERROR}$	36	44	42	22
A8	58	D7	92	$\overline{FLT}$	28	45	48	35
A9	59	D8	90	HLDA	3	46	57	41
A10	60	D9	89	HOLD	4	47	69	49
A11	61	D10	88	INTR	40		71	50
A12	62	D11	87	$\overline{LOCK}$	26		84	63
A13	64	D12	86	M/$\overline{IO}$	23		91	67
A14	65	D13	83	$\overline{NA}$	6		97	68
A15	66	D14	82	NMI	38			77
A16	70	D15	81	PEREQ	37			78
A17	72			$\overline{READY}$	7			85
A18	73			RESET	33			98
A19	74			W/$\overline{R}$	25			
A20	75							
A21	76							
A22	79							
A23	80							

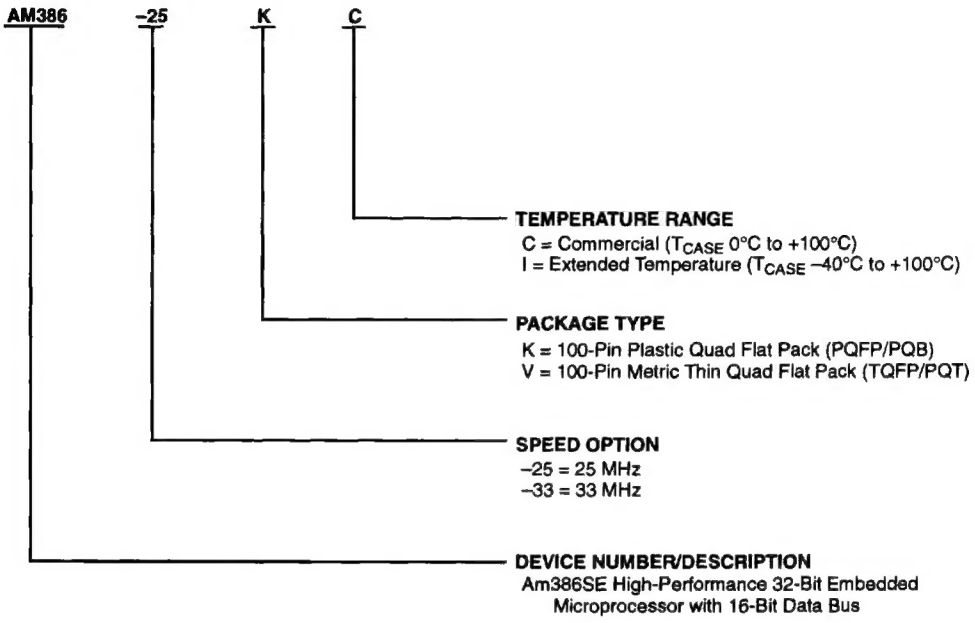
PIN DESIGNATION TABLES (sorted by Pin Number)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	D0	21	V_{CC}	41	V_{SS}	61	A11	81	D15
2	V_{SS}	22	V_{SS}	42	V_{CC}	62	A12	82	D14
3	HLDA	23	M/$\overline{IO}$	43	NC	63	V_{SS}	83	D13
4	HOLD	24	D/$\overline{C}$	44	NC	64	A13	84	V_{CC}
5	V_{SS}	25	W/$\overline{R}$	45	NC	65	A14	85	V_{SS}
6	$\overline{NA}$	26	$\overline{LOCK}$	46	NC	66	A15	86	D12
7	$\overline{READY}$	27	NC	47	NC	67	V_{SS}	87	D11
8	V_{CC}	28	$\overline{FLT}$	48	V_{CC}	68	V_{SS}	88	D10
9	V_{CC}	29	NC	49	V_{SS}	69	V_{CC}	89	D9
10	V_{CC}	30	NC	50	V_{SS}	70	A16	90	D8
11	V_{SS}	31	NC	51	A2	71	V_{CC}	91	V_{CC}
12	V_{SS}	32	V_{CC}	52	A3	72	A17	92	D7
13	V_{SS}	33	RESET	53	A4	73	A18	93	D6
14	V_{SS}	34	$\overline{BUSY}$	54	A5	74	A19	94	D5
15	CLK2	35	V_{SS}	55	A6	75	A20	95	D4
16	$\overline{ADS}$	36	$\overline{ERROR}$	56	A7	76	A21	96	D3
17	$\overline{BLE}$	37	PEREQ	57	V_{CC}	77	V_{SS}	97	V_{CC}
18	A1	38	NMI	58	A8	78	V_{SS}	98	V_{SS}
19	$\overline{BHE}$	39	V_{CC}	59	A9	79	A22	99	D2
20	NC	40	INTR	60	A10	80	A23	100	D1

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of the elements below.



Valid Combinations	
AM386SE	25KC 25KI 25VC 25VI 33KC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations and to check on newly released combinations.

PIN DESCRIPTIONS

A23–A1
Address Bus (Outputs)

Outputs physical memory or port I/O addresses.

$\overline{ADS}$
Address Status (Active Low; Output)

Indicates that a valid bus cycle definition and address (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{BHE}$, $\overline{BLE}$, and A23–A1) are being driven at the Am386SE microprocessor pins.

$\overline{BHE}$, $\overline{BLE}$
Byte Enables (Active Low; Outputs)

Indicate which data bytes of the data bus take part in a bus cycle.

$\overline{BUSY}$
Busy (Active Low; Input)

Signals a busy condition from a processor extension.

CLK2
CLK2 (Input)

Provides the fundamental timing for the Am386SE microprocessor.

D15–D0
Data Bus (Inputs/Outputs)

Inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles.

D/$\overline{C}$
Data/Control (Output)

A bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and code fetch.

$\overline{ERROR}$
Error (Active Low; Input)

Signals an error condition from a processor extension.

$\overline{FLT}$
Float (Active Low; Input)

An input which forces all bidirectional and output signals, including HLDA, to the three-state condition.

HLDA
Bus Hold Acknowledge (Active High; Output)

Output indicates that the Am386SE microprocessor has surrendered control of its logical bus to another bus master.

HOLD
Bus Hold Request (Active High; Input)

Input allows another bus master to request control of the local bus.

INTR
Interrupt Request (Active High; Input)

A maskable input that signals the Am386SE microprocessor to suspend execution of the current program and execute an interrupt acknowledge function.

$\overline{LOCK}$
Bus Lock (Active Low; Output)

A bus cycle definition pin that indicates that other system bus masters are not to gain control of the system bus while it is active.

M/$\overline{IO}$
Memory/IO (Output)

A bus cycle definition pin that distinguishes memory cycles from input/output cycles.

$\overline{NA}$
Next Address (Active Low; Input)

Used to request address pipeline.

NC
No Connect

Should always be left unconnected. Connection of a NC pin may cause the processor to malfunction or be incompatible with future steppings of the Am386SE microprocessor.

NMI
Non-Maskable Interrupt Request
(Active High; Input)

A non-maskable input that signals the Am386SE microprocessor to suspend execution of the current program and execute an interrupt acknowledge function.

PEREQ
Processor Extension Request (Active High; Input)

Indicates that the processor has data to be transferred by the Am386SE microprocessor.

$\overline{READY}$
Bus Ready (Active Low; Input)

Terminates the bus cycle.

RESET
Reset (Active High; Input)

Suspends any operation in progress and places the Am386SE microprocessor in a known reset state.

V_{CC}
System Power (Input)

Provides the 3.0–5 V nominal DC supply input.

V_{SS}
System Ground (Input)

Provides the 0-V connection from which all inputs and outputs are measured.

$W/\overline{R}$
Write/Read (Output)

A bus cycle definition pin that distinguishes write cycles from read cycles.

Am386SE ARCHITECTURE OVERVIEW

The Am386SE microprocessor is code compatible with the Am386DE, 286, and 8086 microprocessors. System manufacturers can provide Am386DE CPU-based embedded controllers optimized for performance and Am386SE CPU-based embedded controllers optimized for cost, both sharing the same operating systems and application software. Systems based on the Am386SE microprocessor can access the world's largest existing microcomputer software base.

Instruction pipelining, high-bus bandwidth, and a very high-performance ALU ensure short average instruction execution times and high system throughput. The Am386SE CPU is capable of execution at sustained rates of 2.5–3.0 million instructions per second (MIPS).

The Am386SE CPU offers on-chip testability and debugging features. Four breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses for powerful debugging of even ROM-based systems.

Base Architecture

The Am386SE microprocessor consists of a central processing unit, a Memory Management Unit, and a bus interface.

The central processing unit consists of the execution unit and the instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction op-codes and stores them in the decoded instruction queue for immediate use by the execution unit.

The segmentation unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The Am386SE microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Address Mode (Protected Mode). In Real Mode the Am386SE CPU operates as a very fast 8086, but with 32-bit extensions, if desired. Real Mode is required primarily to set up the processor for Protected Mode operation.

Finally, to facilitate high-performance system hardware designs, the Am386SE microprocessor bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

Register Set

The Am386SE microprocessor has 30 registers as shown in Figure 1. These registers are grouped into the following seven categories:

General Purpose Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX, and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: The two 32-bit special purpose registers in Figure 1 record or control certain aspects of the Am386SE microprocessor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer (EIP) is 32-bits wide. The EIP controls instruction fetching, and the processor automatically increments it after executing an instruction.

Control Registers: The CR0 32-bit control register is used to control the global nature of the Am386SE microprocessor. The CR0 register contains bits that set the different processor modes (Protected, Real, and Coprocessor Emulation).

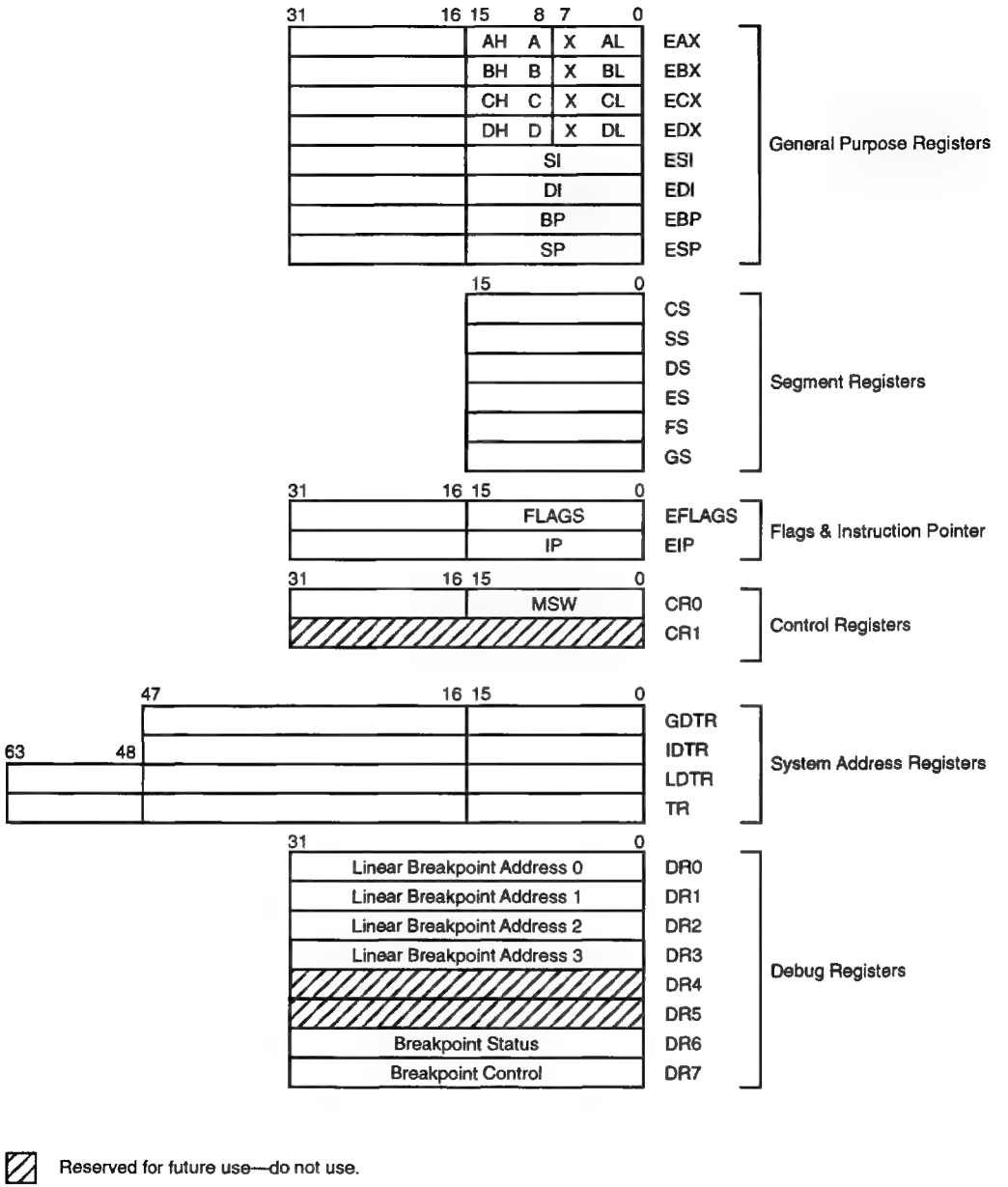


Figure 1. Am386SE Microprocessor Registers

System Address Registers: These four special registers reference the tables or segments supported by the 80286/Am386SE/Am386DE CPU's protection model. These tables or segments are

GDTR (Global Descriptor Table Register)
IDTR (Interrupt Descriptor Table Register)
LDTR (Local Descriptor Table Register)
TR (Task State Segment Register)

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in the section Debugging Support.

EFLAGS Register

The flag register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2, control certain operations and indicate the status of the Am386SE microprocessor. The lower 16 bits (bits 15–0) of EFLAGS contain the 16-bit flag register named FLAGS. This is the default flag register used when executing 8086, 80286, or real mode code. The functions of the flag bits are given in Table 1.

Control Registers

The Am386SE microprocessor has a control register of 32 bits, CR0, to hold the machine state of a global nature. This register is shown in Figure 1 and Figure 2. The defined CR0 bits are described in Table 2.

Instruction Set

The instruction set is divided into nine categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High-Level Language Support
Operating System Support
Processor Control

These instructions are listed in the Instruction Set Clock Count Summary (pages 72 through 86).

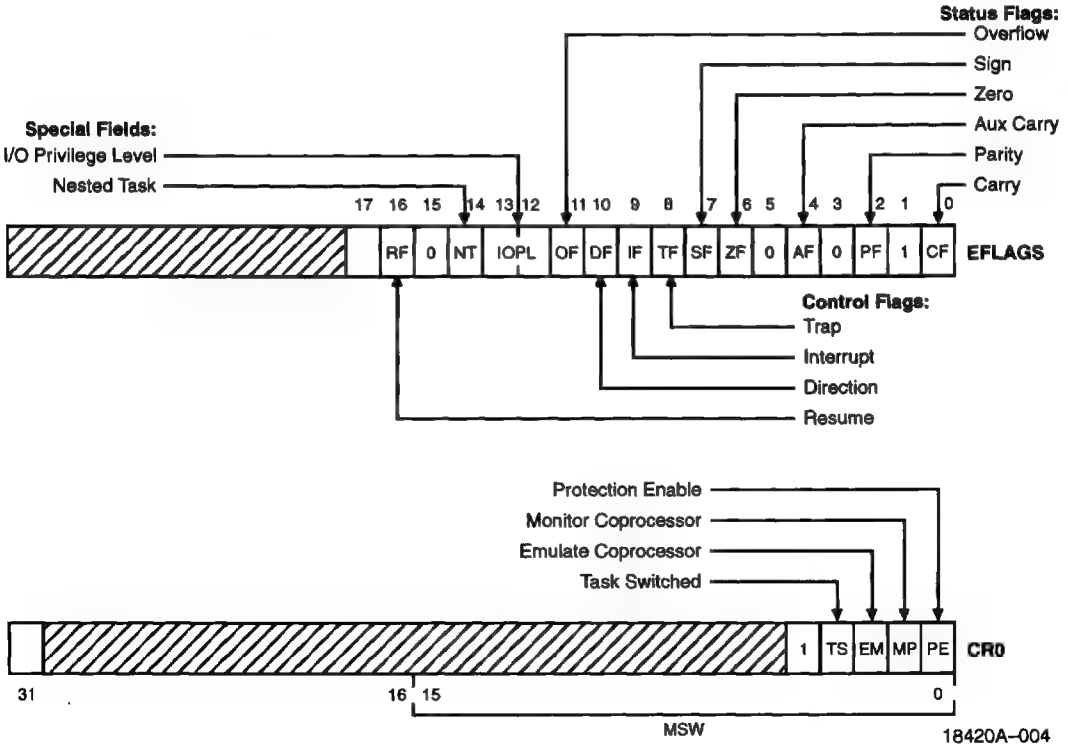


Figure 2. Status and Control Register Bit Functions

Table 1. Flag Definitions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1 bits; cleared otherwise.
4	AF	Auxiliary Carry Flag—Set on carry from or borrow to the low-order 4 bits of AL; cleared otherwise.
6	ZF	Zero Flag—Set if result is zero; cleared otherwise.
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	Single-Step Flag—Once set, a single-step interrupt occurs after the next instruction executes. TF is cleared by the single-step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	Overflow Flag—Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12, 13	IOPL	I/O Privilege Level—Indicates the maximum CPL permitted to execute I/O instructions without generating an Exception 13 fault or consulting the I/O permission bit map while executing in protected mode.
14	NT	Nested Task—Indicates that the execution of the current task is nested within another task.
16	RF	Resume Flag—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction.

Table 2. CR0 Definitions

Bit Position	Name	Function
0	PE	Protection Mode Enable—Places the Am386SE microprocessor into protected mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by loading CR0; it cannot be reset by the LMSW instruction.
1	MP	Monitor Coprocessor Extension—Allows WAIT instructions to cause a processor extension Not Present exception (number 7).
2	EM	Emulate Processor Extension—Causes a processor extension Not Present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task Switched—Indicates the next instruction using a processor extension will cause Exception 7, allowing software to test whether the current processor extension context belongs to the current task.

All Am386SE microprocessor instructions operate on either 0, 1, 2, or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instruction generally is two bytes long. The average instruction is 3.2 bytes long. Since the Am386SE CPU has a 16-byte prefetch instruction queue, an average of five instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Immediate to Register
Memory to Memory
Register to Memory
Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the Am386SE microprocessor (32-bit code), operands are 8 or 32 bits; when executing existing 8086 or 80286 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

Memory Organization

Memory on the Am386SE microprocessor is divided into 8-bit quantities (Bytes), 16-bit quantities (Words), and 32-bit quantities (Dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a Word or Dword is the byte address of the low-order byte.

In addition to these basic data types, the Am386SE microprocessor supports a larger unit of memory in segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. The Am386SE CPU supports segmentation in order to provide maximum flexibility to the system designer. Segmentation is useful for organizing memory in logical modules, and, as such, is a tool for the application programmer.

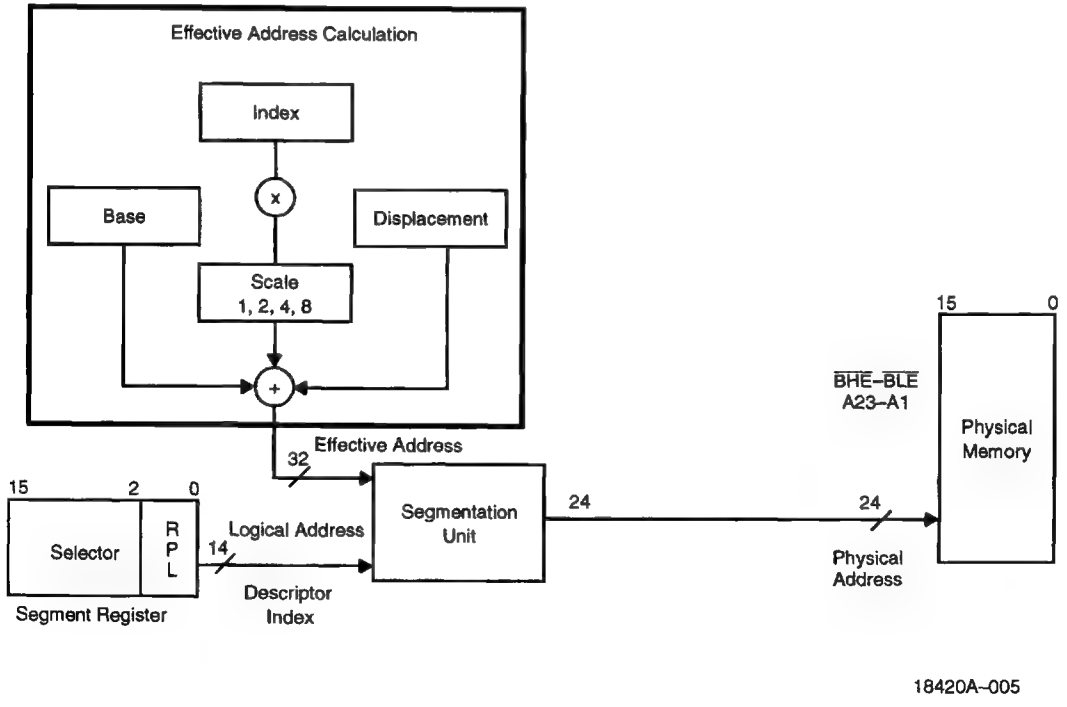


Figure 3. Address Translation

Address Spaces

The Am386SE microprocessor has two types of address spaces: logical and physical. A logical address (also known as a virtual address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (Base, Index, Displacement) discussed in the section Addressing Modes, into an effective address. This effective address, along with the selector, is known as the logical address. Each task on the Am386SE CPU has a maximum of 16K (2^{14}–1) selectors.

The segmentation unit translates the logical address space into a 32-bit physical address space. The 32-bit physical address is then truncated into a 24-bit physical address. The physical address is what appears on the address pins. The total address space of the Am386SE CPU is 16 Mbytes.

The primary differences between Real Mode and Protected Mode are how the segmentation unit performs the translation of the logical address into the physical address and the size of the address space. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the effective address to form the physical address. This physical address is limited to 1 Mbyte.

In protected mode, every selector has a logical base address associated with it that can be up to 32 bits in length. This 32-bit logical base address is added to the effective address to form a final 32-bit physical address. This address reflects physical memory and is truncated so that only the lower 24 bits of this address are used to address the 16-Mbyte memory address space.

Figure 3 shows the relationship between the various address spaces.

Segment Register Usage

The main data structure used to organize memory is the segment. On the Am386SE CPU, segments are variable sized blocks of physical addresses which have certain attributes associated with them. There are two main types of segments, code and data. The segments are of variable size and can be as small as 1 byte or as large as 16 Mbytes.

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. The segment register is automatically chosen according to the rules of Table 3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; stack references use the SS register; and, instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register and override the implicit rules listed in Table 3. The override prefixes also allow the use of the ES, FS, and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all six segments could have the base address set to zero and create a system with 16-Mbyte physical address space. This creates a system where the virtual address space is the same as the physical address space. Further details of segmentation are discussed in the section Protected Mode Architecture on page 24.

Addressing Modes

The Am386SE microprocessor provides a total of eight addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high-level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

Register and Immediate Modes

Two of the addressing modes provide for instructions that operate on register or immediate operands.

Register Operand Mode: The operand is located in one of the 8-, 16-, or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the op-code.

32-bit Memory Addressing Modes

The remaining six modes provide a mechanism for specifying the effective address of an operand. The physical address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see Figure 3).

Displacement: an 8-, 16-, or 32-bit immediate value, following the instruction.

Base: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

Index: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4, or 8. The scaled index is especially useful for accessing arrays or structures.

Table 3. Segment Register Selection Rules

Type Of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET Instructions	SS	None
Destination of STOS, MOVE, REP, STOS, REP MOVS Instructions	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	CS, SS, ES, FS, GS
[EBX]	DS	CS, SS, ES, FS, GS
[ECX]	DS	CS, SS, ES, FS, GS
[EDX]	DS	CS, SS, ES, FS, GS
[ESI]	DS	CS, SS, ES, FS, GS
[EDI]	DS	CS, SS, ES, FS, GS
[EBP]	SS	CS, SS, ES, FS, GS
[ESP]	SS	CS, SS, ES, FS, GS

Combinations of these three components make up the six additional addressing modes.

1. **Direct Mode**: The operand's offset is contained as part of the instruction as an 8-, 16-, or 32-bit displacement.
2. **Register Indirect Mode**: A Base register contains the address of the operand.
3. **Based Mode**: A Base register's contents are added to a Displacement to form the operand's offset.
4. **Scaled Index Mode**: An Index register's contents are multiplied by a Scaling factor, and the result is added to a Displacement to form the operand's offset.
5. **Based Scaled Index Mode**: The contents of an Index register are multiplied by a Scaling factor, and the result is added to the contents of a Base register to obtain the operand's offset.
6. **Based Scaled Index Mode with Displacement**: The contents of an Index register are multiplied by a Scaling factor, and the result is added to the contents of a Base register and a Displacement to form the operand's offset.

As shown in Figure 4, the Effective Address (EA) of an operand is calculated according to the following formula:

$$EA = Base_{Register} + (Index_{Register} \times Scaling) + Displacement$$

There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

Differences Between 16- and 32-bit Addresses

In order to provide software compatibility with the 8086 and the 80286, the Am386SE microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in a Segment Descriptor. If the D bit is 0, then all operand lengths and effective addresses are assumed to be16-bits long. If the D bit is 1, then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16 bits.

Regardless of the default precision of the operands or addresses, the Am386SE microprocessor is able to execute either 16 or 32-bit instructions. This is specified through the use of override prefixes. Two prefixes, the Operand Length Prefix and the Address Length Prefix, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by assemblers.

The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds 0FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional Am386SE CPU addressing modes.

When executing 32-bit code, the Am386SE CPU uses either 8 or 32-bit displacements, and any register can be used as Base or Index registers. When executing 16-bit code, the displacements are either 8 or 16-bits, and the Base and Index registers conform to the 80286 model. Table 4 illustrates the differences.

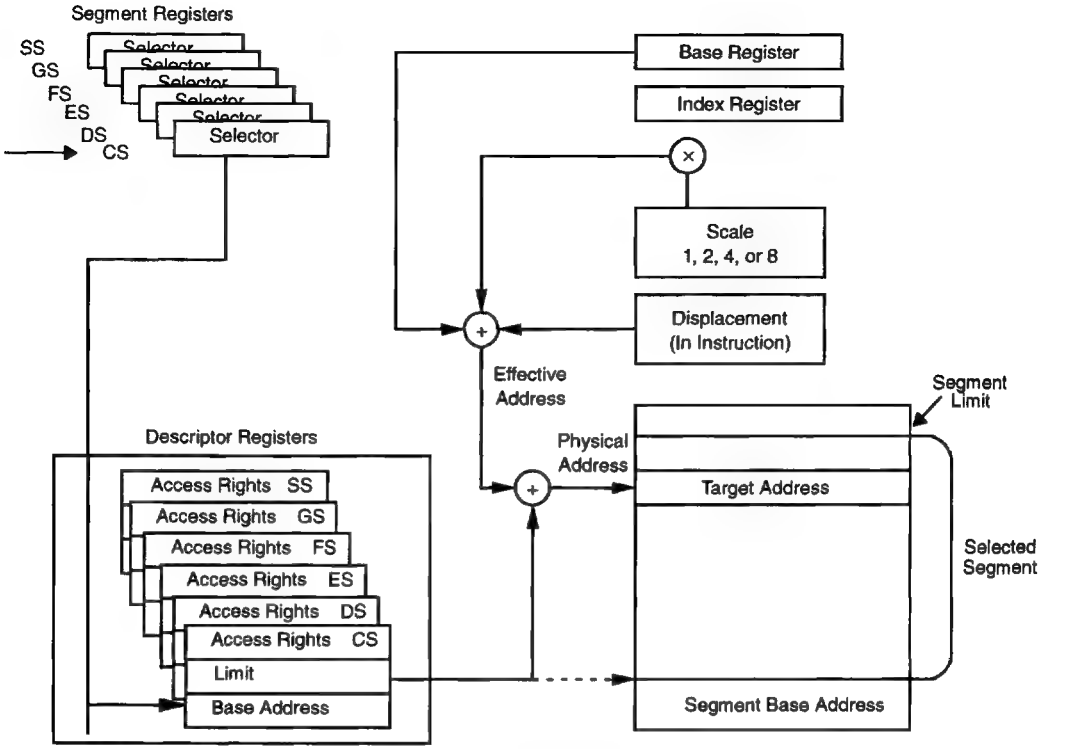


Figure 4. Addressing Mode Calculations 18420A–006

Data Types

The Am386SE microprocessor supports all of the data types commonly used in high-level languages.

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.

Bit String: A set of contiguous bits; on the Am386SE microprocessor, bit strings can be up to 4 Gbits long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Dword): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Dword): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Pointer: A 16- or 32-bit offset-only quantity which indirectly references another memory location.

Long Pointer: A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.

Char: A byte representation of an ASCII alphanumeric or control character.

String: A contiguous sequence of bytes, Words, or Dwords. A string may contain between 1 byte to 4 Gbytes.

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the Am386SE microprocessor is coupled with a 387SX math coprocessor, the following common floating-point types are supported.

Floating Point: A signed 32-, 64-, or 80-bit real number representation. Floating-point numbers are supported by 387SX-compatible math coprocessors.

Table 4. Base and Index Registers for 16- and 32-bit Addresses

	16-Bit Addressing	32-Bit Addressing
Base Register	BX, BP	Any 32-bit GP Register
Index Register	SI, DI	Any 32-bit GP Register
Scale Factor	None	1, 2, 4, 8
Displacement	0, 8, 16 bits	0, 8, 32 bits

Figure 5 illustrates the data types supported by the Am386SE microprocessor and a 387SX compatible math coprocessor.

I/O Space

The Am386SE CPU has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space, although the Am386SE CPU also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes which can be divided into 64K 8-bit ports or 32K 16-bit ports, or any combination of ports which add up to no more than 64 Kbytes. The 64-Kbyte I/O address space refers to physical addresses since I/O instructions do not go through the segmentation hardware. The M/$\overline{IO}$ pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed by the In and Out instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/$\overline{IO}$ pin to be driven Low. I/O port addresses 00F8H through 00FFH are reserved for future use.

Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow in order to handle external events, report errors, or report exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT n instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately after the interrupted instruction.

Exceptions are classified as faults, traps, or aborts, depending on the way they are reported and whether or not restart of the instruction causing the exception is supported. Faults are exceptions that are detected and serviced *before* the execution of the faulting instruction. Traps are exceptions that are reported immediately *after* the execution of the instruction which caused the problem. Aborts are exceptions that do not permit the precise location of the instruction causing the exception to be determined.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point to the instruction causing the exception and will include any leading instruction prefixes. Table 5 summarizes the possible interrupts for the Am386SE microprocessor and shows where the return address points.

The Am386SE CPU has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode, the vectors are 4-byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8-byte quantities which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for future use and the remaining 224 are free to be used by the system designer.

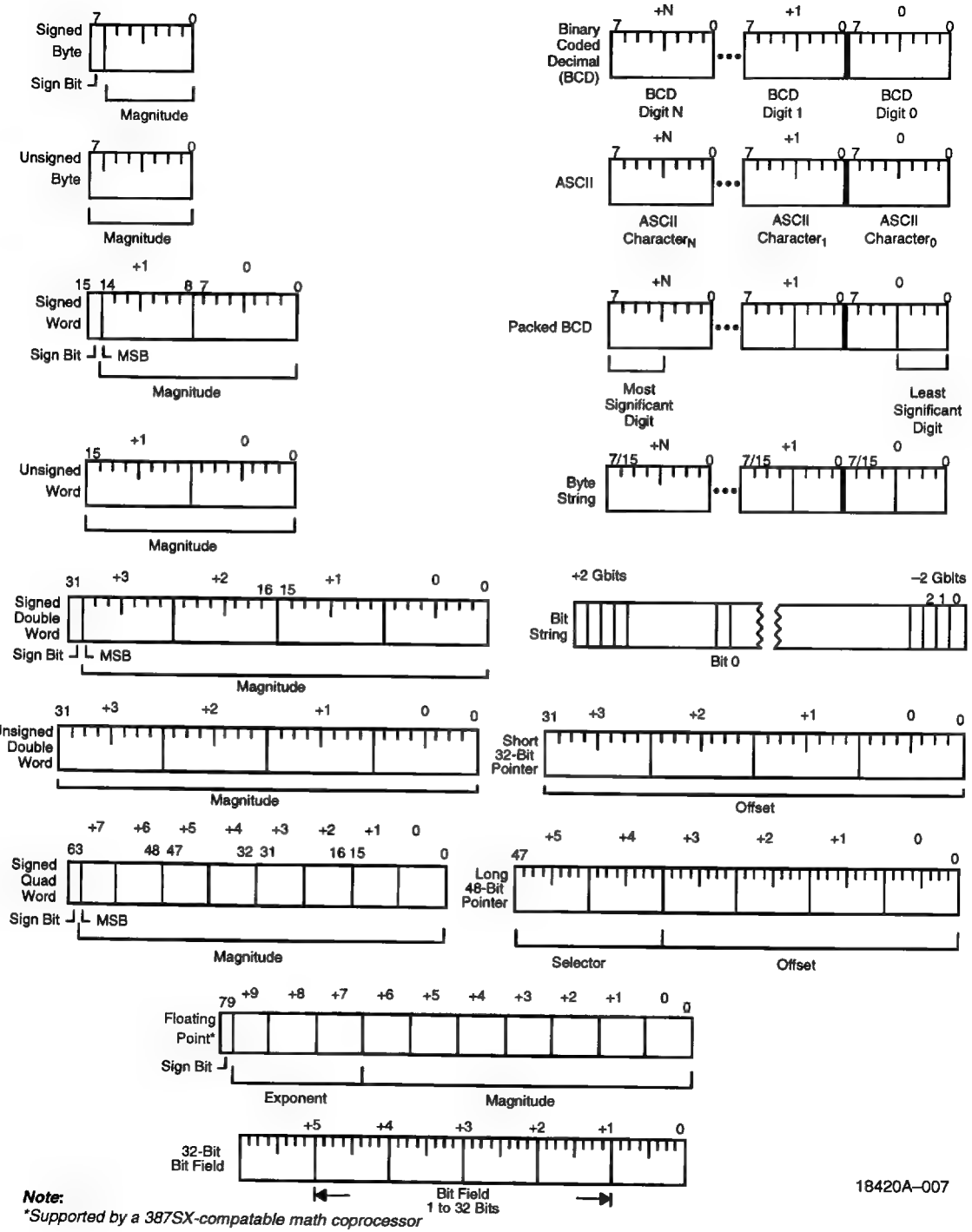
Interrupt Processing

When an interrupt occurs, the following actions happen. First, the current program address and Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the Am386SE microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the Am386SE microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

Maskable interrupts are the most common way to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled High and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an interrupt window between memory moves that allows interrupts during long string moves).



Note:
**Supported by a 387SX-compatable math coprocessor*

Figure 5. Am386SE Microprocessor Supported Data Types

Table 5. Interrupt Vector Assignments

Function	Interrupt Number	Instructions That Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	Yes	FAULT
Debug Exception	1	Any Instruction	Yes	TRAP
NMI Interrupt	2	INT2 or NMI	No	NMI
One Byte Interrupt	3	INT	No	TRAP
Interrupt on Overflow	4	INTO	No	TRAP
Array Bounds Check	5	BOUND	Yes	FAULT
Invalid Op-code	6	Any Illegal Instruction	Yes	FAULT
Device Not Available	7	ESC, WAIT	Yes	FAULT
Double Fault	8	Any instruction that can generate an exception		ABORT
Coprocessor Segment Overrun	9	ESC	No	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	Yes	FAULT
Segment Not Present	11	Segment Register Instructions	Yes	FAULT
Stack Fault	12	Stack References	Yes	FAULT
General Protection Fault	13	Any Memory References	Yes	FAULT
Reserved for Future Use	14			
Coprocessor Error	16	ESC, WAIT	Yes	FAULT
Reserved for Future Use	17–32			
Two Byte Interrupt	0–255	INT n	No	TRAP

Note: *Some debug exceptions may report both traps on the previous instruction and faults on the next instruction.*

When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Interrupts through interrupt gates automatically reset IF bit, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGs register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled High it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the Am386SE microprocessor will not service any further NMI request or INT requests until an Interrupt Return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

Software Interrupts

A third type of interrupt/exception for the Am386SE CPU is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single-step interrupt. It is discussed in the section Single-Step Trap on page 22.

Interrupt and Exception Priorities

Interrupts are externally generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are both recognized at the same instruction boundary, the Am386SE microprocessor invokes the NMI service routine first. If maskable interrupts are still enabled after the NMI service routine has been invoked, then the

Am386SE CPU will invoke the appropriate interrupt service routine.

Exception Checking Sequence

As the Am386SE microprocessor executes instructions, it follows a consistent cycle in checking for exceptions. Consider the case of the Am386SE microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (Exceptions 11 and 13).
5. Check for Faults decoding the next instruction (Exception 6 if illegal op-code; Exception 6 if in Real Mode and attempting to execute an instruction for Protected Mode only; or Exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0)).
6. If WAIT op-code, check if TS=1 and MP=1(Exception 7 if both are 1).
7. If ESCape op-code for math coprocessor, check if EM=1 or TS=1 (Exception 7 if either is 1).
8. If WAIT op-code or ESCape op-code for math coprocessor, check $\overline{\text{ERROR}}$ input signal (Exception 16 if $\overline{\text{ERROR}}$ input is asserted).
9. Check each memory reference required by the instruction for segmentation faults that prevent transferring the entire memory quantity (Exceptions 11, 12, and 13).

Instruction Restart

The Am386SE microprocessor fully supports restarting all instructions after Faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in the previous Exception Checking Sequence), the Am386SE microprocessor invokes the appropriate exception service routine.

Double Fault

A Double Fault (Exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12, or 13), but in the process of doing so detects an exception.

Reset and Initialization

When the processor is initialized or Reset, the registers have the values shown in Table 6. The Am386SE CPU will then start executing instructions near the top of physical memory, at location 0FFFFF0H. When the first intersegment Jump or Call is executed, address lines A23–A20 will drop Low for CS-relative memory cycles, and the Am386SE CPU will only execute instructions in the lower 1 Mbyte of physical memory. This allows the system designer to use a shadow ROM at the top of physical memory to initialize the system and take care of Resets.

Reset forces the Am386SE microprocessor to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350, and 450 CLK2 periods after Reset becomes inactive, the Am386SE microprocessor will start executing instructions at the top of physical memory.

Table 6. Register Values after Reset

Register	Reset Value	Notes
Flag Word (EFLAGS)	uuuu0002H	1
Machine Status Word (CR0)	uuuuuu10H	
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	2
Data Segment (DS)	0000H	3
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	3
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX Register	0000H	4
EDX Register	Component and Stepping ID	5
All Other Registers	Undefined	6

Notes:

1. EFLAGS Register. The upper 14 bits of the EFLAGS register are undefined; all defined flag bits are zero.
2. The Code Segment register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
3. The Data and Extra Segment registers (DS and ES) will have their Base Address set to 000000000H and Limit set to 0FFFFH.
4. If self-test is selected, the EAX register should contain a 0 value. If a value of 0 is not found, the self-test has detected a flaw in the part.
5. EDX register always holds a component and stepping identifier.
6. All undefined bits are reserved for future use and should not be used.

Testability

The Am386SE microprocessor, like the Am386DE microprocessor, offers testability features that include a self-test and direct access to the page translation cache.

Self-Test

The Am386SE microprocessor has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the Am386SE CPU can be tested during self-test.

Self-Test is initiated on the Am386SE microprocessor when the Reset pin transitions from High to Low, and the $\overline{\text{BUSY}}$ pin is Low. The self-test takes about 2^{20} clocks. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX are zero. If the results of the EAX are not zero, then the self-test has detected a flaw in the part.

Debugging Support

The Am386SE microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint op-code (0CCH).
2. The single-step capability provided by the TF bit in the flag register.
3. The code and data breakpoint capability provided by the Debug Registers DR3–DR0, DR6, and DR7.

Breakpoint Instruction

A single-byte software interrupt (INT 3) breakpoint instruction is available for use by software debuggers. The breakpoint op-code is 0CCH, and generates an Exception 3 trap when executed.

Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAGS register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto-vectored to Exception 1.

Debug Registers

The Debug Registers are an advanced debugging feature of the Am386SE microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint op-code.

The Am386SE microprocessor contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the Debug Registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to Exception 1. Figure 6 shows the breakpoint status and control registers.

REAL MODE ARCHITECTURE

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the Am386SE microprocessor. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286.

The default operand size in Real Mode is 16 bits, as in the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the Am386SE microprocessor in Real Mode is 64 Kbytes, so 32-bit addresses must have a value less than 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode operation.

Memory Addressing

Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a 20-bit physical address or a 1-Mbyte address space. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64-Kbytes long, and may be read, written, or executed. The Am386SE microprocessor will generate an Exception 13 if a data operand or instruction fetch occurs past the end of a segment.

Reserved Locations

There are two fixed areas in memory that are reserved in Real Address Mode: the system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations 0FFFFF0H through 0FFFFFFH are reserved for system initialization.

Interrupts

Many of the exceptions discussed in the Interrupts and Exceptions section on page 20, are not applicable to Real Mode operation; in particular, Exceptions 10 and 11 do not occur in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 7 identifies these exceptions.

Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, $\overline{FLT}$, INTR with interrupts enabled (IF=1), or Reset will force the Am386SE microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shut-down can occur under two conditions:

1. An interrupt or an exception occurs (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table.
2. A Call, INT, or Push instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 000FH) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). Otherwise, shutdown can only be exited by a processor reset.

Table 7. Exceptions in Real Mode

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT vector is not within table limit.	Before Instruction
CS, DS, ES, FS, GS Segment Overrun exception	13	Word memory reference with offset = 0FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment Overrun exception	12	Stack Reference beyond offset = 0FFFFH.	Before Instruction

LOCK Operation

The only instruction forms where the LOCK prefix is legal on the Am386SE microprocessor are shown in Table 8.

The LOCK prefix is not supported during repeat string instructions.

Table 8. Legal Instructions for the LOCK Prefix

Op-Code	Operands (Dest. Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/Immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB AND, SUB, XOR	Mem, Reg/Immed
NOT, NEG, INC, DEC	Mem

An Exception 6 will be generated if a LOCK prefix is placed before any instruction form or op-code not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above.

The LOCK prefix is not IOPL-sensitive on the Am386SE microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed in Table 8.

PROTECTED MODE ARCHITECTURE

The complete capabilities of the Am386SE microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the physical address space to 4 Gb (2^{32} bytes). In addition, Protected Mode allows the Am386SE CPU to run all of the existing Am386DE CPU (using only 16 Mb of physical memory), 80286, and 8086 CPU's software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions specially optimized for supporting multitasking operating systems. The base architecture of the Am386SE microprocessor remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's viewpoint is the increased address space and a different addressing mechanism.

Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address: a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit physical address. The address is then used as a 24-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode, the selector is used to specify an index into an operating system defined table (see Figure 7). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

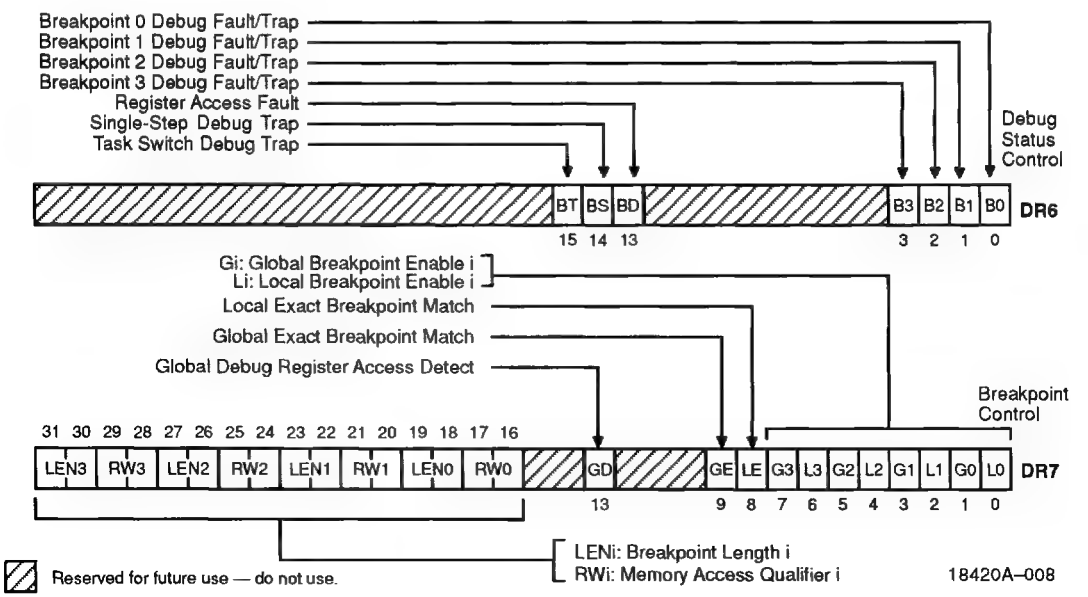


Figure 6. Debug Registers

Segmentation

Segmentation is a method of memory management. It provides the basis for software protection and is used to encapsulate regions of memory that have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment is stored in an 8-byte data structure called a descriptor. All of the descriptors in a system are contained in descriptor tables which are recognized by hardware.

Terminology

The following terms are used throughout the discussion of descriptors, privilege levels, and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the least two significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and the DPL. EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

Descriptor Tables

The descriptor tables define all of the segments which are used in an Am386SE microprocessor system. There are three types of tables which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and Interrupt Descriptor Table. All of the tables are variable length memory arrays and can vary in size from 8 bytes to 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit physical base address and the 16-bit limit of each table.

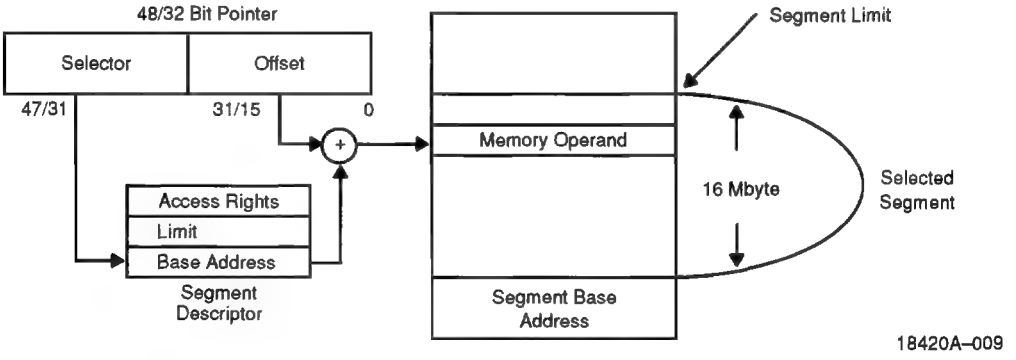


Figure 7. Protected Mode Addressing

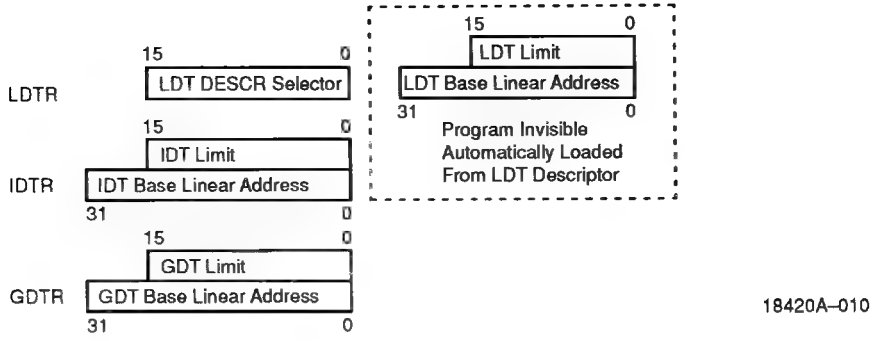


Figure 8. Descriptor Table Registers

Each of the tables has a register associated with it: GDTR, LDTR, and IDTR (see Figure 8). The LGDT, LLDT, and LIDT instructions load the base and limit of the Global, Local, and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These are privileged instructions.

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every Am386SE CPU system contains a GDT.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see Figure 1).

Interrupt Descriptor Table

The third table needed for Am386SE microprocessor systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of the up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 interrupts reserved for future use. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

Descriptors

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of physical address space. These attributes include the 32-bit base physical address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write, or execute privileges, the default size of the operands (16 bit or 32 bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 9 shows the general format of a descriptor. All segments on the Am386SE microprocessor have three attribute fields in common: the P bit, the DPL bit, and the S bit. The P (Present) Bit is 1 if the segment is loaded in physical memory. If P=0, then any attempt to access this segment causes a Not Present exception (number 11). The Descriptor Privilege Level (DPL) is a two bit field which specifies the protection level, 0–3, associated with a segment.

The Am386SE microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment bit (S) determines if a given segment is a system segment or a code or data segment. If the S bit is 1, then the segment is either a code or data segment; if it is 0, then the segment is a system segment.

Code and Data Descriptors (S=1)

Figure 10 shows the general format of a code and data descriptor, and Table 9 illustrates how the bits in the Access Right Byte are interpreted.

Code and data segments have several descriptor fields in common. The accessed bit (A) is set whenever the processor accesses a descriptor. The granularity bit (G) specifies if a segment length is byte-granular or page-granular.

System Descriptor Formats (S=0)

System segments describe information about operating system tables, task, and gates. Figure 11 shows the general format of system segment descriptors, and the various types of system segments. Am386SE CPU system descriptors (which are the same as Am386DE CPU system descriptors) contain a 32-bit base physical address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

Differences Between Am386SE Microprocessor and 80286 Descriptors

In order to provide operating system compatibility with the 80286, the Am386SE CPU supports all of the 80286 segment descriptors. The 80286 system segment descriptors contain a 24-bit base address and 16-bit limit, while the Am386SE CPU system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Am386SE CPU call gates.

Table 9. Access Rights Byte Definition for Code and Data Descriptors

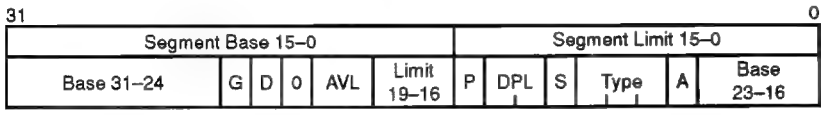
Bit Position	Name	Function	
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists. Base and Limit are not used.	
6–5	Descriptor Privilege Levels (DPL)	Segment privilege attribute used in privilege tests.	
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) Segment Descriptor S = 0 System Segment Descriptor or Gate Descriptor	
3	Executable (E)	E = 0 Descriptor type is data segment.	If Data Segment (S = 1, E = 0)
2	Expansion Direction (ED)	ED = 0 Expand up segment offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.	
1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.	
3	Executable (E)	E = 1 Descriptor type is code segment.	If Data Segment (S = 1, E = 1)
2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.	
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL), as shown in Figure 12. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

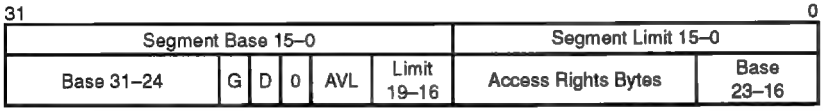
In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with the selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.



Base	Base Address of the segment	A	Accessed Bit
Limit	The length of the segment	G	Granularity Bit (1 = Segment length in page-granular; 0 = Segment length is byte-granular)
P	Present Bit (1 = Present; 0 = Not Present)		
DPL	Descriptor Privilege Level 0–3	D	Default Operation Size (recognized in code segment descriptors only; 1 = 32-bit segment, 0 = 16-bit segment)
S	Segment Descriptor (0 = System Descriptor; 1 = Code or Data Segment Descriptor)	0	Bit must be zero for compatibility with future processors
Type	Type of Segment	AVL	Available field for user or OS

18420A–011

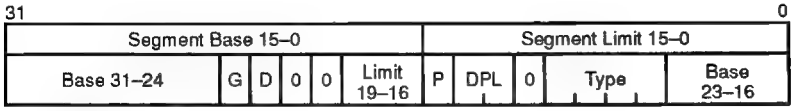
Figure 9. Segment Descriptors



D/B	1 = Default Instruction Attributes are 32 bits 0 = Default Instruction Attributes are 16 bits	G	Granularity Bit	1 = Segment length is page-granular 0 = Segment length in byte-granular
AVL	Available field for user or OS	0	Bit must be zero for compatibility with future processors	

18420A–012

Figure 10. Code and Data Descriptors



Type	Definition	Type	Definition
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available TSS
2	LDT	A	Undefined (Reserved)
3	Busy 80286 TSS	B	Busy TSS
4	80286 Call Gate	C	Am386SE CPU Call Gate
5	Task Gate (for 80286 or Am386SE CPU Task)	D	Undefined (Reserved)
6	80286 Interrupt Gate	E	Am386SE CPU Interrupt Gate
7	80286 Trap Gate	F	Am386SE CPU Trap Gate

Figure 11. System Descriptors 18420A–013

Protection

The Am386SE microprocessor has four levels of protection which are optimized to support a multitasking operating system and to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors.

The four-level hierarchical privilege system is an extension of the user/supervisor privilege mode commonly used by minicomputers. The Privilege Levels (PL) are numbered 0 through 3. Level 0 is the most privileged level.

Rules of Privilege

The Am386SE microprocessor controls access to both data and procedures between levels of a task, according to the following rules:

- Data stored in a segment with privilege level p can be accessed only by code executing at a privilege level at least as privileged as p.
- A code segment/procedure with privilege level p can only be called by a task executing at the same or lesser privilege level than p.

Privilege Levels

At any point in time, a task on the Am386SE microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL=3 may call an operating system routine at PL=1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's Effective Privilege Level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any IRPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

I/O Privilege

The I/O Privilege Level (IOPL) lets the operating system code executing at CPL=0 define the least privileged level at which I/O instructions can be used. An Exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged then the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an Exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI, and LOCK prefix.

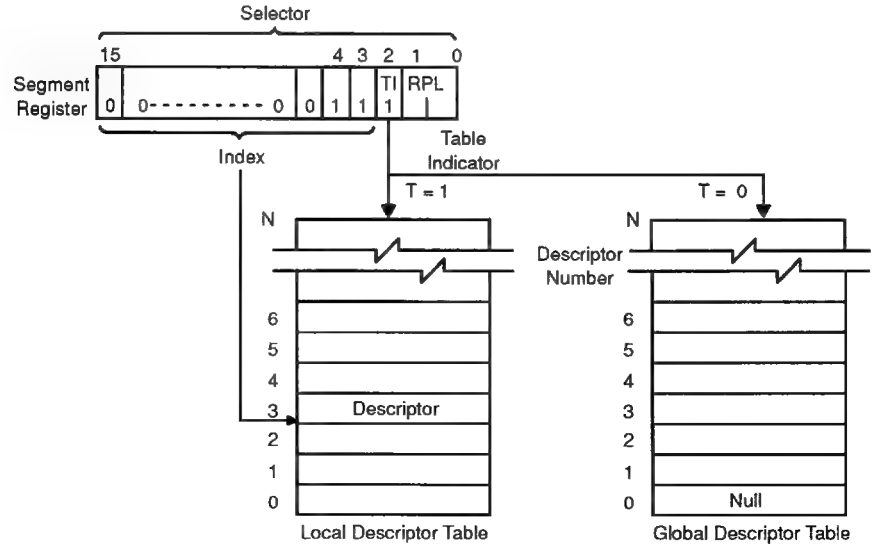


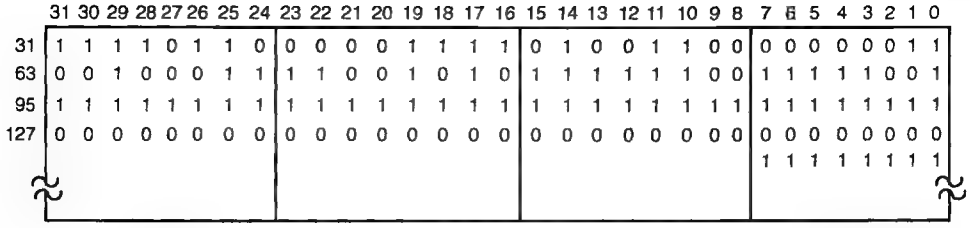
Figure 12. Example Descriptor Selection

Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used, and CPL, RPL, and DPL as described above.

Any time an instruction loads a data segment register (DS, ES, FS, GS) the Am386SE CPU makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

Finally, the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an Exception 13 (General Protection Fault) is generated.



I/O Ports Accessible: 2 → 9, 12, 13, 15, 20 → 24, 27, 33, 34, 40, 41, 48, 50, 52, 53, 58 → 60, 62, 63, 96 → 127

18420A–016

Figure 13. Sample I/O Permission Bit Map

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL of all other descriptor types or a privilege level violation will cause an Exception 13. A stack not present fault causes an Exception 12.

Privilege Level Transfers

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 10. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an Exception 13.

CALL Gates

Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level	CALL	Call Gate	GDT/LDT
Interrupt within task may change CPL	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

Notes:
**NT (Nested Task bit of flag register) = 0*
***NT (Nested Task bit of flag register) = 1*

Task Switching

A very important attribute of any multitasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The Am386SE microprocessor directly supports this operation by providing a task switch instruction in hardware. The task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an intersegment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 14) containing the entire execution state. A task gate descriptor contains a TSS selector. The Am386SE microprocessor supports both 80286 and Am386SE CPU TSSs. The limit of an Am386SE CPU TSS must be greater than 64H (2BH for a 80286 TSS), and can be as large as 16 Mbytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, the time the task has spent running, or open files belonging to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Am386SE microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TSS descriptor are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The currently executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which is useful to the operating system. The Nested Task bit (NT) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return. If NT = 1, IRET performs a task switch operation base to the previous task. The NT bit is set or reset in the following fashion:

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT (the NT bit will be restored after execution of the interrupt handler). NT may also be set or cleared by POPF or IRET instructions.

The Am386SE microprocessor task state segment is marked busy by changing the descriptor type field from Type 9 to Type 0BH. An 80286 TSS is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes an Exception 13.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit (TS) in the CR0 register helps deal with the coprocessor's state in a multitasking environment. Whenever the Am386SE microprocessor switches tasks, it sets the TS bit. The Am386SE CPU detects the first use of a processor extension instruction after a task switch and causes the processor extension Not Available Exception 7. The exception handler for Exception 7 may then decide whether to save the state of the coprocessor.

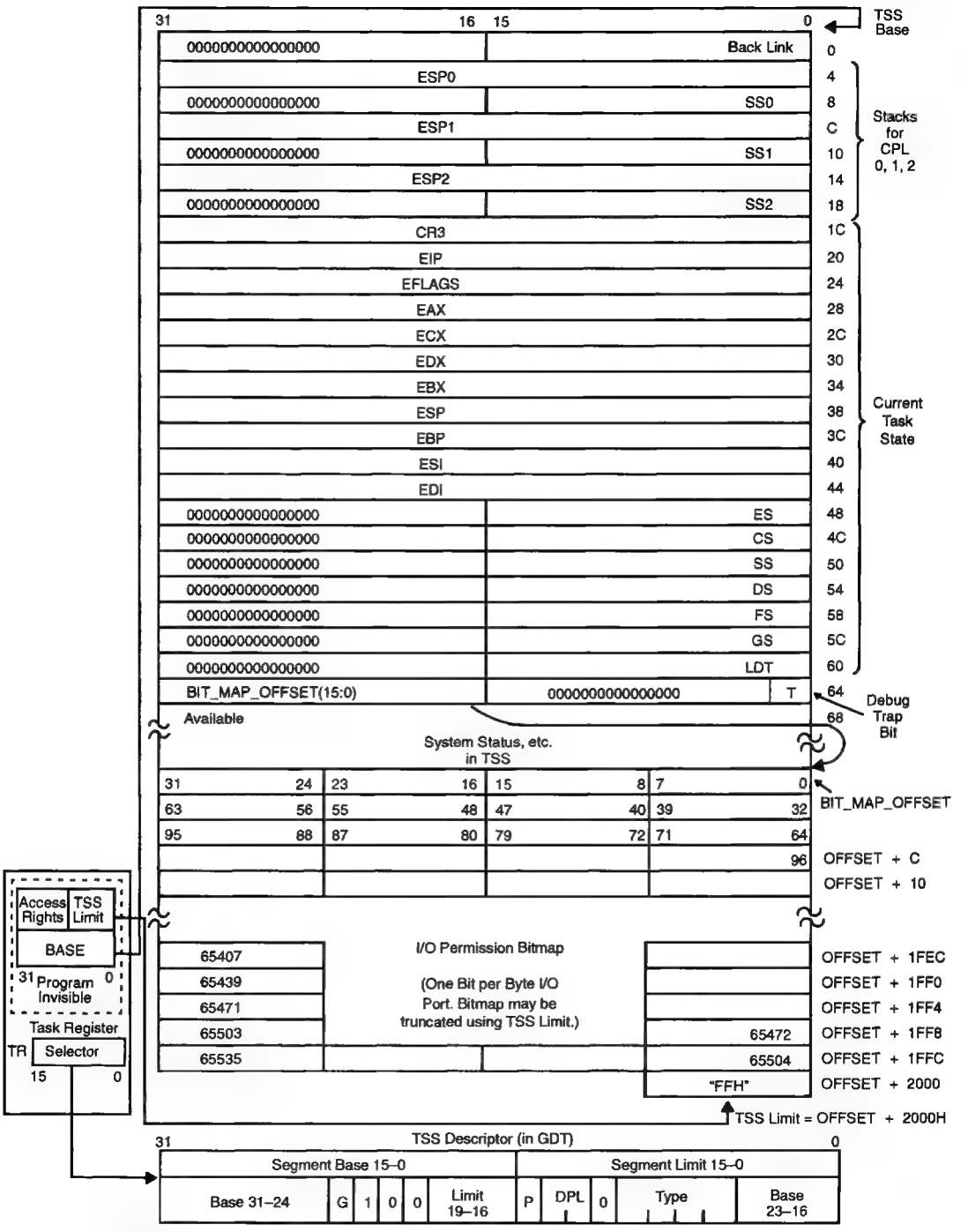
The T bit in the Am386SE microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T=1, then upon entry to a new task a debug Exception 1 will be generated.

Initialization and Transition To Protected Mode

Since the Am386SE microprocessor begins executing in Real Mode immediately after RESET, it is necessary to initialize the system tables and registers with the appropriate values. The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and the GDT must contain descriptors for the initial code and data segments.

Protected Mode is enabled by loading CR0 with PE bit set. This can be accomplished by using the MOV CR0, R/M instruction. After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode is to use the built-in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor.



Type = 9: Available TSS
Type = 8: Busy TSS

18420A–015

Figure 14. TSS and TSS Registers

FUNCTIONAL DATA

The Am386SE microprocessor features a straightforward functional interface to the external hardware (see Figure 15). The Am386SE CPU has separate parallel buses for data and address. The data bus is 16-bits in width, and bidirectional. The address bus outputs 24-bit address values using 23 address lines and two Byte Enable signals.

The Am386SE microprocessor has two selectable address bus cycles: address pipelined and non-address pipelined. The address pipelining option allows as much time as possible for data access by starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor CLK cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. The Am386SE microprocessor bus cycles perform data transfer in a minimum of only two clock periods. The maximum transfer bandwidth at 16 MHz is therefore 16 Mbyte/s. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgment of the cycle.

The Am386SE microprocessor can relinquish control of its local buses to allow mastership by other devices, such as Direct Memory Access (DMA) channels. When relinquished, HLDA is the only output pin driven by the Am386SE microprocessor, providing near complete isolation of the processor from its system (all other output pins are in a float condition).

Signal Description Overview

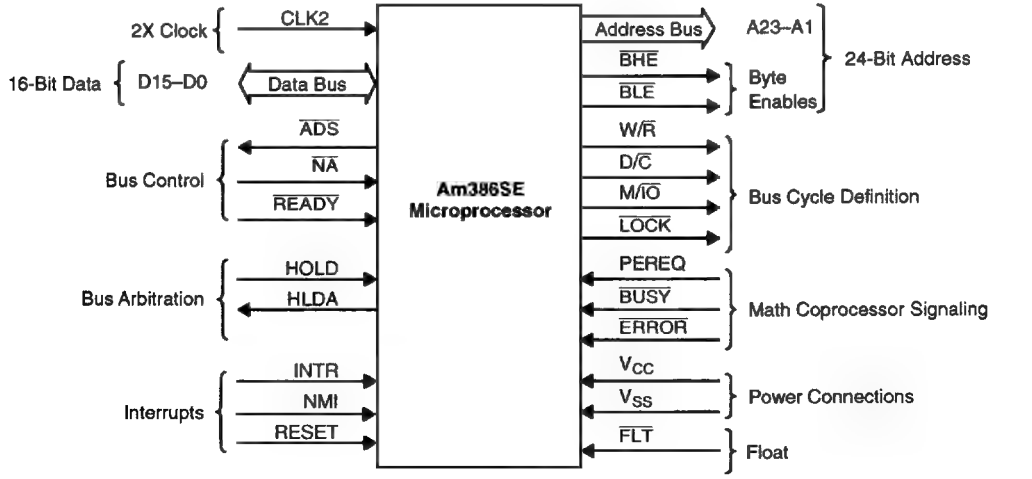
Below is a brief description of the Am386SE microprocessor input and output signals arranged by functional groups.

Example signal: M/$\overline{IO}$–High voltage indicates memory selected
–Low voltage indicates I/O selected

The signal descriptions sometimes refer to Switching timing parameters, such as t25 Reset Setup Time and t26 Reset Hold Time. The values of these parameters can be found in the Switching Characteristics table.

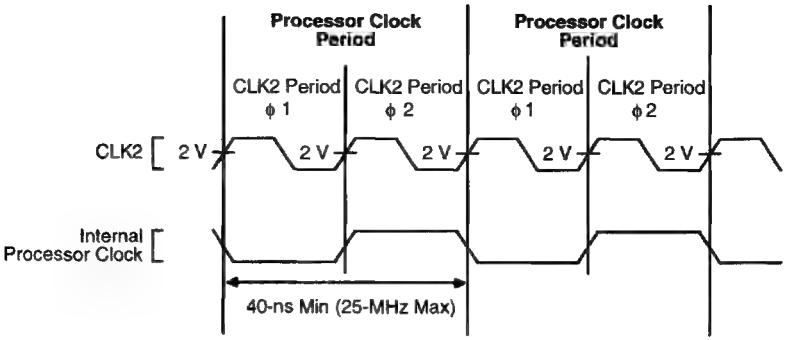
Clock (CLK2)

CLK2 provides the fundamental timing for the Am386SE microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, phase one and phase two. Each CLK2 period is a phase of the internal clock. Figure 16 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times, t25 and t26.



18420A–017

Figure 15. Functional Signal Groups



18420A–018

Figure 16. CLK2 Signal and Internal Processor Clock

Data Bus (D15–D0)

These three-state, bidirectional signals provide the general purpose data path between the Am386SE microprocessor and other devices. The data bus outputs are active High and will float during Bus Hold Acknowledge. Data bus reads require that read-data setup and hold times (t21 and t22) be met relative to CLK2 for correct operation.

Address Bus (A23–A1, $\overline{BHE}$, $\overline{BLE}$)

These three-state outputs provide physical memory addresses or I/O port addresses. A23–A16 are Low during I/O transfers, except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A22–A16 are driven Low and A23 is driven High, so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the Am386SE microprocessor for coprocessor commands is 8000F8H, the I/O addresses driven by the Am386SE CPU for coprocessor data are 8000FCH or 8000FEH or cycles to a 387SX math coprocessor.

The address bus is capable of addressing 16 Mbytes of physical memory space (000000H through FFFFFFH), and 64 Kbytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active High and will float during Bus Hold Acknowledge.

The Byte Enable outputs, $\overline{BHE}$ and $\overline{BLE}$, directly indicate which bytes of the 16-bit data bus are involved with the current transfer. $\overline{BHE}$ applies to D15–D8 and $\overline{BLE}$ applies to D7–D0. If both $\overline{BHE}$ and $\overline{BLE}$ are asserted, then 16 bits of data are being transferred. See Table 11 for a complete decoding of these signals. The Byte Enables are active Low and will float during Bus Hold Acknowledge.

Table 11. Byte Enable Definitions

$\overline{BHE}$	$\overline{BLE}$	Function
0	0	Word Transfer
0	1	Byte transfer on upper byte of the data bus, D15–D8
1	0	Byte transfer on lower byte of the data bus, D7–D0
1	1	Never occurs

Bus Cycle Definition Signals (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$)

These three-state outputs define the type of bus cycle being performed: W/$\overline{R}$ distinguishes between write and read cycles; D/$\overline{C}$ distinguishes between data and control cycles; M/$\overline{IO}$ distinguishes between memory and I/O cycles; and, $\overline{LOCK}$ distinguishes between locked and unlocked bus cycles. All of these signals are active Low and will float during Bus Acknowledge.

The primary bus cycle definition signals are W/$\overline{R}$, D/$\overline{C}$, and M/$\overline{IO}$, since these are the signals driven valid as $\overline{ADS}$ (Address Status output) becomes active. The $\overline{LOCK}$ is driven valid at the same time the bus cycle begins, which, due to address pipelining, could be after $\overline{ADS}$ becomes active. Exact bus cycle definitions, as a function of W/$\overline{R}$, D/$\overline{C}$, and M/$\overline{IO}$, are given in Table 12.

$\overline{LOCK}$ indicates that other system bus masters are not to gain control of the system bus while it is active. $\overline{LOCK}$ is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when $\overline{READY}$ is returned at the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when $\overline{READY}$ is returned in a previous bus cycle and another is pending ($\overline{ADS}$ is active), or the clock in which $\overline{ADS}$ is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be

Table 12. Bus Cycle Definition

M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	Bus Cycle Type	Locked?
0	0	0	Interrupt Acknowledge	Yes
0	0	1	Does not occur	—
0	1	0	I/O Data Read	No
0	1	1	I/O Data Write	No
1	0	0	Memory Code Read	No
1	0	1	Halt: Address = 2, $\overline{BHE}$ = 1, $\overline{BLE}$ = 0 — Shutdown: Address = 0, $\overline{BHE}$ = 1, $\overline{BLE}$ = 0	No
1	1	0	Memory Data Read	Some Cycles
1	1	1	Memory Data Write	Some Cycles

locked longer than desired. The $\overline{LOCK}$ signal may be explicitly activated by the LOCK prefix on certain instructions.

$\overline{LOCK}$ is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

Bus Control Signals ($\overline{ADS}$, $\overline{READY}$, $\overline{NA}$)

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining and bus cycle termination.

***Address Status* ($\overline{ADS}$)**

This three-state output indicates that a valid bus cycle definition and address (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{BHE}$, $\overline{BLE}$, and (A23–A1) are being driven at the Am386SE microprocessor pins. $\overline{ADS}$ is an active Low output. Once $\overline{ADS}$ is driven active, valid address, Byte Enables, and definition signals will not change. In addition, $\overline{ADS}$ will remain active until its associated bus cycle begins (when $\overline{READY}$ is returned for the previous bus cycle when running pipelined bus cycles). When address pipelining is utilized, maximum throughput is achieved by initiating bus cycles when $\overline{ADS}$ and $\overline{READY}$ are active in the same clock cycle. $\overline{ADS}$ will float during Bus Hold Acknowledge. See sections Non-Pipelined Address and Pipelined Address for additional information on how $\overline{ADS}$ is asserted for different bus states.

***Transfer Acknowledge* ($\overline{READY}$)**

This input indicates the current bus cycle is complete, and the active bytes indicated by $\overline{BHE}$ and $\overline{BLE}$ are accepted or provided. When $\overline{READY}$ is sampled active during a read cycle or interrupt acknowledge cycle, the Am386SE microprocessor latches the input data and terminates the cycle. When $\overline{READY}$ is sampled active during a write cycle, the processor terminates the bus cycle.

$\overline{READY}$ is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. $\overline{READY}$ must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, $\overline{READY}$ must always meet setup and hold times (t19 and t20) for correct operation.

***Next Address Request* ($\overline{NA}$)**

This is used to request address pipelining. This input indicates the system is prepared to accept new values of $\overline{BHE}$, $\overline{BLE}$, A23–A1, W/$\overline{R}$, D/$\overline{C}$, and M/$\overline{IO}$ from the Am386SE CPU even if the end of the current cycle is not being acknowledged on $\overline{READY}$. If this input is active when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. $\overline{NA}$ is ignored in CLK cycles in which $\overline{ADS}$ or $\overline{READY}$ is activated. This signal is active Low and must satisfy setup and hold times (t15 and t16) for correct operation. See the sections Read and Write Cycles on page 43, and Pipelined Address on page 46 for additional information.

Bus Arbitration Signals (HOLD, HLDA)

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See the section Entering and Exiting Hold Acknowledge on page 51 for additional information.

***Bus Hold Request* (*HOLD*)**

This input indicates some device other than the Am386SE microprocessor requires bus master-ship. When control is granted, the Am386SE CPU floats A23–A1, $\overline{BHE}$, $\overline{BLE}$, D15–D0, $\overline{LOCK}$, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, and $\overline{ADS}$, and then activates HLDA, thus entering the Bus Hold Acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the Am386SE microprocessor will deactivate HLDA and drive the local bus (at the same time), thus terminating the Hold Acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the Hold Acknowledge (HLDA) state, since none of the Am386SE microprocessor floated outputs have internal pull-up resistors. See the section

Resistor Recommendations, on page 59 for additional information. HOLD is not recognized while RESET is active. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the Hold Acknowledge (high impedance) state.

HOLD is a level-sensitive, active High, synchronous input. HOLD signals must always meet setup and hold times (t23 and t24) for correct operation.

Bus Hold Acknowledge (HLDA)

When active (High), this output indicates the Am386SE microprocessor has relinquished control of its local bus in response to an asserted HOLD signal, and is in the Bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near complete signal isolation. In the HLDA state is the only signal being driven by the Am386SE microprocessor. The other output or bidirectional signals (D15–D0, $\overline{BHE}$, $\overline{BLE}$, A23–A1, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, and $\overline{ADS}$) are in a high-impedance state so the requesting bus master may control them. These pins remain Off throughout the time that HLDA remains active (see Table 13). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See the section Resistor Recommendations on page 59 for additional information.

When the HOLD signal is made inactive, the Am386SE microprocessor will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 13. Output Pin State During HOLD

Pin Value	Pin Names
1	HLDA
Float	$\overline{LOCK}$, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{ADS}$, A23–A1, $\overline{BHE}$, $\overline{BLE}$, D15–D0

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware fault-tolerant applications.

HOLD Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the $\overline{LOCK}$ signal (internal to the CPU) activated by the LOCK prefix, and interrupts. The Am386SE microprocessor will not honor a HOLD request until the current bus operation is complete.

The Am386SE microprocessor breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the $\overline{LOCK}$ signal is not asserted. The Am386SE microprocessor breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again, the $\overline{LOCK}$ signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The Am386SE microprocessor will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must ensure that $\overline{READY}$ returns promptly.

Coprocessor Interface Signals (PEREQ, $\overline{BUSY}$, $\overline{ERROR}$)

In the following sections are descriptions of signals dedicated to the math coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, the following signals control communication between the Am386SE microprocessor and its 387SX math coprocessor extension.

Coprocessor Request (PEREQ)

When asserted (High), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the Am386SE microprocessor. In response, the Am386SE microprocessor transfers information between the math coprocessor and memory. Because the Am386SE CPU has internally stored the math coprocessor op-code being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is level-sensitive active High asynchronous signal. Setup and hold times (t29 and t30) relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 Kohms to Ground so that it will not float active when left unconnected.

Coprocessor Busy ($\overline{BUSY}$)

When asserted Low, this input indicates that the coprocessor is still executing an instruction, and is not yet able to accept another. When the Am386SE microprocessor encounters any coprocessor instruction which operates on the numerics stack (e.g., load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the $\overline{BUSY}$ input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT, FNSTENV, FNSAVE, FNSTSW, FNSTCW, and FNCLEX coprocessor instructions are allowed to execute even if $\overline{BUSY}$ is active, since these instructions are used for coprocessor initialization and exception-clearing.

$\overline{BUSY}$ is an active Low, level-sensitive, asynchronous signal. Setup and hold times (t29 and t30), relative to the CLK2 signal, must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 Kohms to V_{CC} so that it will not float active when left unconnected.

BUSY serves an additional function. If BUSY is sampled Low at the falling edge of RESET, the Am386SE microprocessor performs an internal self-test (see the section Bus Activity During and Following Reset on page 53). If BUSY is sampled High, no self-test is performed.

***Coprocessor Error* (ERROR)**

When asserted Low, this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the Am386SE microprocessor when a coprocessor instruction is encountered, and if active, the Am386SE CPU generates Exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the Am386SE CPU generating Exception 16 even if ERROR is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV, and FNSAVE.

ERROR is an active Low, level-sensitive, asynchronous signal. Setup and hold times (t29 and t30), relative to the CLK2 signal, must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 Kohms to V_{CC} so that it will not float active when left unconnected.

Interrupt Signals (INTR, NMI, RESET)

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the Am386SE microprocessor Flag Register IF bit. When the Am386SE CPU responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D7–D0 to identify the source of the interrupt.

INTR is an active High, level-sensitive, asynchronous signal. Setup and hold times (t27 and t28), relative to the CLK2 signal, must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction in the Am386SE microprocessor's Execution Unit. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the instruction. If recognized, the Am386SE CPU will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI)

This input indicates a request for interrupt service which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is an active High, rising edge-sensitive, asynchronous signal. Setup and hold times (t27 and t28), relative to the CLK2 signal, must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the instruction boundary in the Am386SE microprocessor's Execution Unit.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is reasserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, an INTR request will not be recognized until interrupts are re-enabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the Am386SE microprocessor encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the Am386SE microprocessor's Execution Unit except for the following cases:
   - —Repeat string instructions can be interrupted after each iteration.
   - —If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP. This allows the entire stack pointer to be loaded without interruption.
   - —If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

   The longest latency occurs when the interrupt request arrives while the Am386SE microprocessor is executing a long instruction such as multiplication, division, or a task switch in the Protected Mode.
4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the Am386SE microprocessor.

Reset

This input signal suspends any operation in progress and places the Am386SE microprocessor in a known

reset state. The Am386SE CPU is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins, except $\overline{FLT}$, are ignored, and all other bus pins are driven to an Idle Bus state, as shown in Table 14. If RESET and HOLD are both active at a point in time, RESET takes priority even if the Am386SE microprocessor was in a Hold Acknowledge state prior to RESET active.

Reset is an active High, level-sensitive, synchronous signal. Setup and hold times (t25 and t26) must be met in order to assure proper operation of the Am386SE microprocessor.

Bus Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte and word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two physical bus cycles are performed as required for unaligned operand transfers.

The Am386SE microprocessor address signals are designed to simplify external system hardware. Higher-order address bits are provided by A23–A1. $\overline{BHE}$ and $\overline{BLE}$ provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs $\overline{BHE}$ and $\overline{BLE}$ are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 15.

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See the section Bus Functional Description on page 39.

Table 14. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
$\overline{ADS}$	1
D15–D0	Float
$\overline{BHE}$, $\overline{BLE}$	0
A23–A1	1
W/$\overline{R}$	0
D/$\overline{C}$	1
M/$\overline{IO}$	0
$\overline{LOCK}$	1
HLDA	0

Table 15. Byte Enables and Associated Data and Operand Bytes

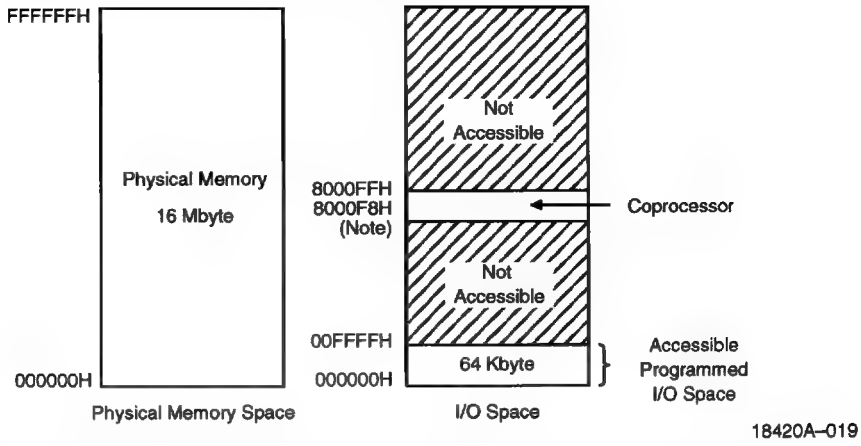
Byte Enable Signal	Associated Data Bus
$\overline{BLE}$	D7–D0 (Byte 0—least significant
BHE	D15–D8 (Byte 1—most significant)

Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, I/O-mapped, or both. As shown in Figure 17, physical memory addresses range from 000000H to 0FFFFFFH (16 Mb) and I/O addresses from 000000H to 00FFFFH (64 Kb). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A23 and M/$\overline{IO}$ signals.

Bus Functional Description

The Am386SE microprocessor has separate, parallel buses for data and address. The data bus is 16-bits in width and is bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 23 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.



Note:
Since A23 is High during automatic communication with coprocessor, A23 High and $\overline{M/IO}$ Low can be used to easily generate a coprocessor select signal.

Figure 17. Physical Memory and I/O Spaces

The definition of each bus cycle is given by three signals: M/$\overline{IO}$, W/$\overline{R}$, and D/$\overline{C}$. At the same time, a valid address is present on the Byte Enable signals, $\overline{BHE}$ and $\overline{BLE}$, and the other address signals, A23–A1. A status signal, $\overline{ADS}$, indicates when the Am386SE microprocessor issues a new bus cycle definition and address.

Collectively, the address bus, data bus, and all associated control signals are referred to simply as the bus. When active, the bus performs one of the bus cycles below:

1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space
5. Read from I/O space (or math coprocessor)
6. Write to I/O space (or math coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

Table 12 shows the encoding of the bus cycle definition signals for each bus cycle. See the Bus Cycle Definition Signals section, on page 35 for additional information.

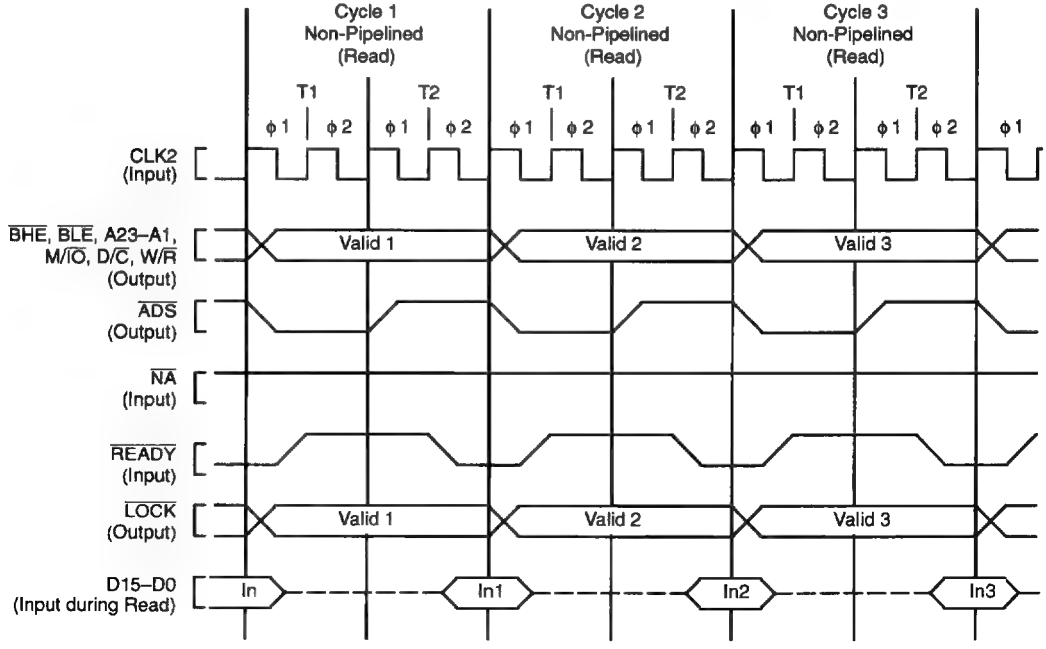
When the Am386SE microprocessor bus is not performing one of the activities listed above, it is either idle or in the Hold Acknowledge state, which may be detected externally. The idle state can be identified by the Am386SE CPU giving no further assertions on its address strobe output ($\overline{ADS}$) since the beginning of its most recent cycle, and the most recent bus cycle having been terminated. The Hold Acknowledge state is identified by the Am386SE microprocessor asserting its Hold Acknowledge (HLDA) output.

Bus Functional Description

The Am386SE microprocessor has separate, parallel buses for data and address. The data bus is 16-bits in width and is bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

The fastest Am386SE microprocessor bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown in Figure 18. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.



Note:
Fastest non-pipelined cycles consist of T1 and T2.

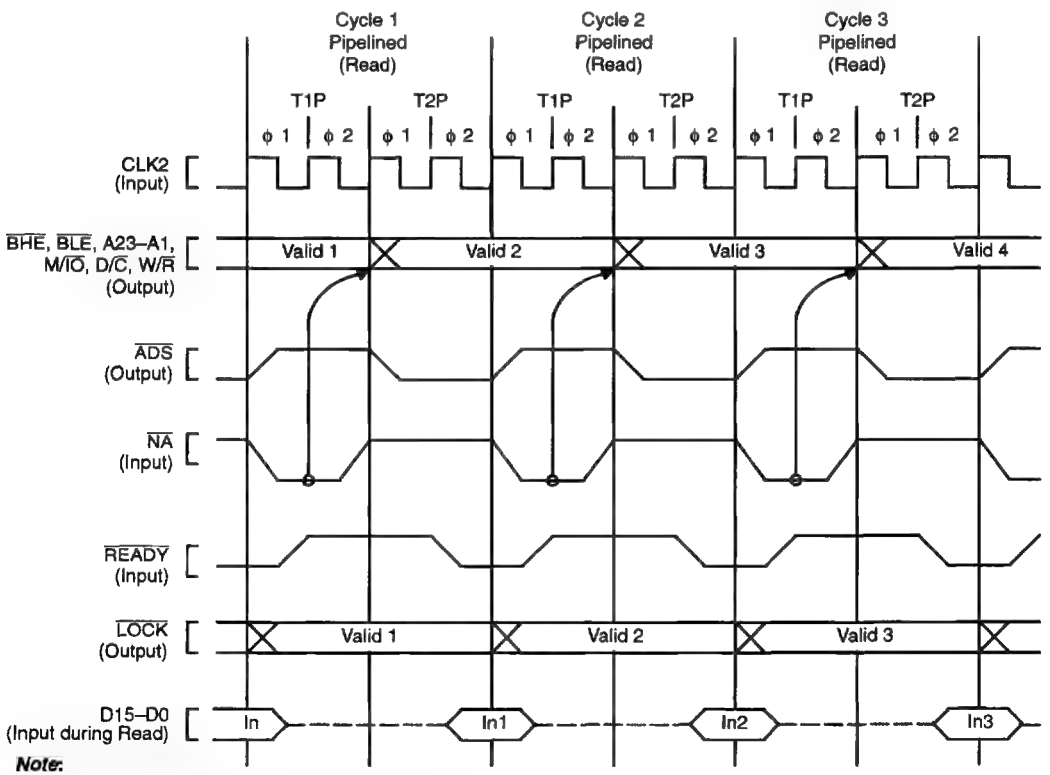
Figure 18. Fastest Read Cycles with Non-Pipelined Address Timing 18420A–020

Every bus cycle continues until it is acknowledged by the external system hardware, using the Am386SE microprocessor $\overline{READY}$ input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If $\overline{READY}$ is not immediately asserted however, T2 states are repeated indefinitely until the $\overline{READY}$ input is sampled active.

The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address ($\overline{NA}$) input.

When address pipelining is selected, the address ($\overline{BHE}$, $\overline{BLE}$, and A23–A1) and definition (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, and $\overline{LOCK}$) of the next cycle are available before the end of the current cycle. To signal their availability, the Am386SE microprocessor address status output ($\overline{ADS}$) is asserted. Figure 19 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 19 the fastest bus cycles using pipelined address require only two bus states, named T1P and T2P. Therefore, cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.



Note:
Fastest pipelined bus cycles consist of T1P and T2P.

Figure 19. Fastest Read Cycles with Pipelined Address Timing 18420A–021

Read and Write Cycles

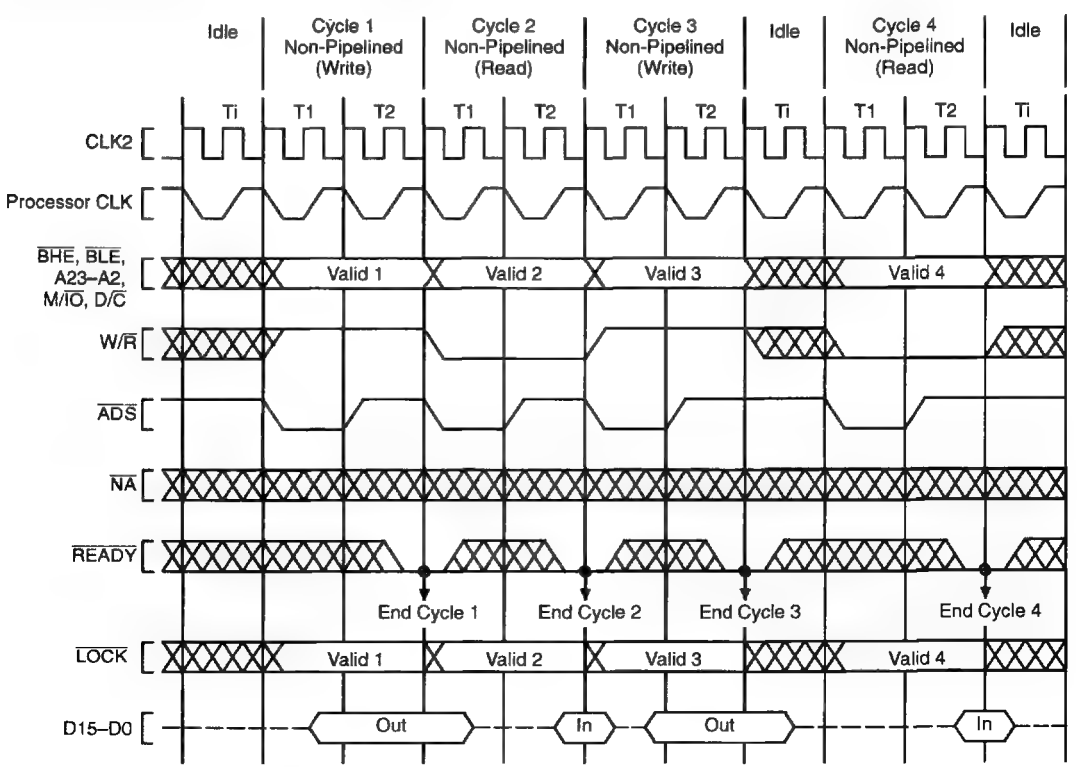
Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.

Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined address timing. However, the $\overline{NA}$ (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the Am386SE microprocessor has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by $\overline{READY}$.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the $\overline{READY}$ input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type, asserts the $\overline{READY}$ input at the appropriate time.

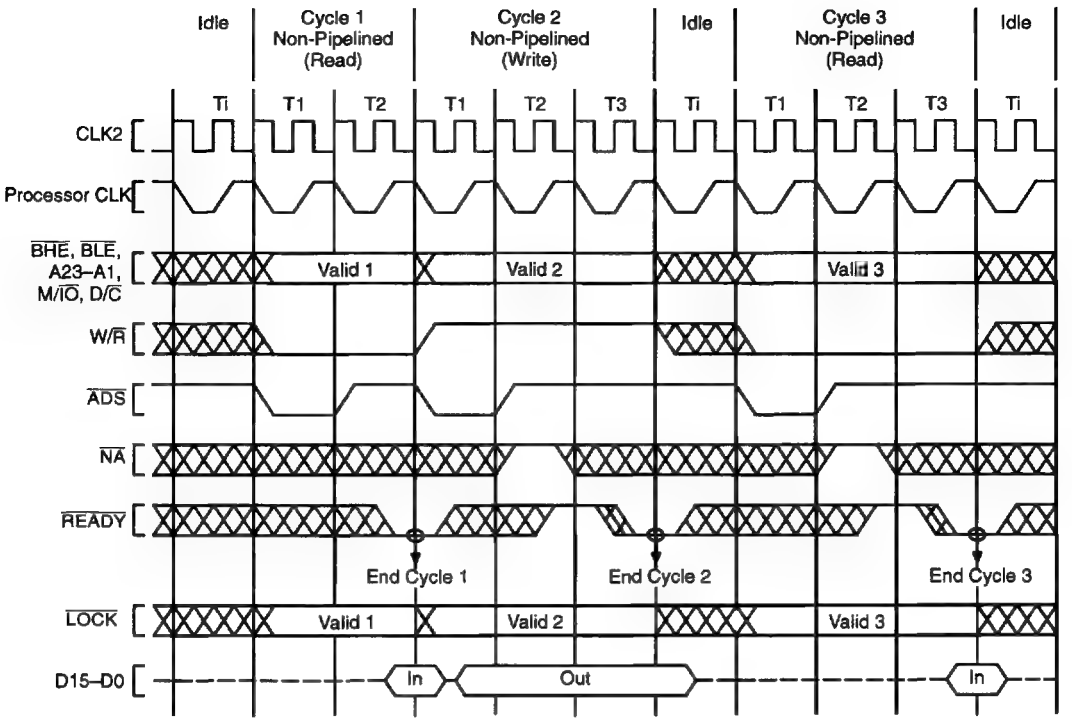
At the end of the second bus state within the bus cycle, $\overline{READY}$ is sampled. At that time, if external hardware acknowledges the bus cycle by asserting $\overline{READY}$, the bus cycle terminates as shown in Figure 20. If $\overline{READY}$ is negated, as in Figure 21, the Am386SE microprocessor executes another bus state (a wait state) and $\overline{READY}$ is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by $\overline{READY}$ asserted.

When the current cycle is acknowledged, the Am386SE microprocessor terminates it. When a read cycle is acknowledged, the Am386SE CPU latches the information present at its data pins. When a write cycle is acknowledged, the Am386SE microprocessor's write data remains valid throughout phase one of the next bus state, to provide write data hold time.



Note:
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state; an active bus cycle can immediately follow the write cycle.

Figure 20. Various Bus Cycles with Non-Pipelined Address (Zero Wait States) 18420A–022



Note:
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state; an active bus cycle can immediately follow the write cycle.

18420A–023

Figure 21. Various Bus Cycles with Non-Pipelined Address (Various Number of Wait States)

Non-Pipelined Address

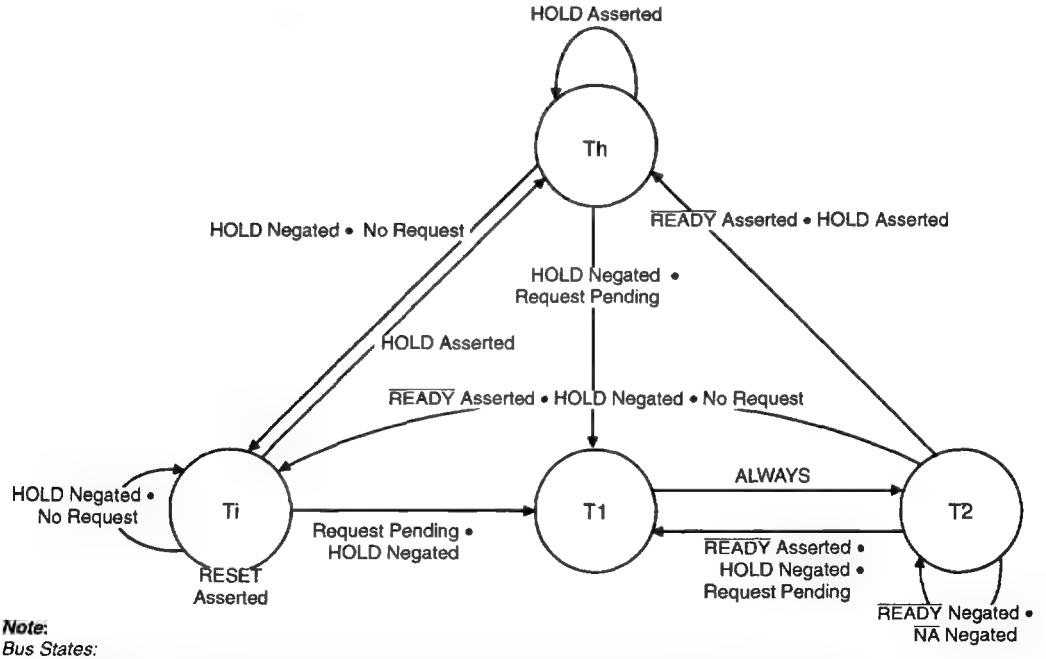
Any bus cycle may be performed with non-pipelined address timing. For example, Figure 20 shows a mixture of read and write cycles with non-pipelined address timing. Figure 20 shows that the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe ($\overline{ADS}$) is simultaneously asserted.

During read or write cycles the data bus behaves as follows. If the cycle is a read, the Am386SE microprocessor floats its data signal to allow driving by the external device being addressed. The Am386SE microprocessor requires that all data bus pins be at a valid logic state (High or Low) at the end of each read cycle, when $\overline{READY}$ is asserted. The system **must** be designed to meet this requirement. If the cycle is a write, data signals are driven by the Am386SE CPU beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

Figure 21 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. $\overline{READY}$ is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore, Cycles 2 and 3 have T2 repeated again. At the end of the second T2, $\overline{READY}$ is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added, and it is desirable to maintain non-pipelined address timing, it is necessary to negate $\overline{NA}$ during each T2 state, except the last one, as shown in Figure 21, Cycles 2 and 3. If $\overline{NA}$ is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

The bus states and transitions, when address pipelining is not used, are completely illustrated by Figure 22. The bus transitions between four possible states, T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, Ti, or in the Hold Acknowledge state Th.



Note:
Bus States:
T1—First clock of a non-pipelined bus cycle (Am386SE CPU drives new address and asserts $\overline{ADS}$).
T2—Subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
Ti—Idle state.
Th—Hold Acknowledge state (Am386SE CPU asserts HLDA).

The fastest bus cycle consists of two states: T1 and T2.
Four basic states describe bus operation when not using pipelined address.

18420A–024

Figure 22. Bus States (Not Using Pipelined Address)

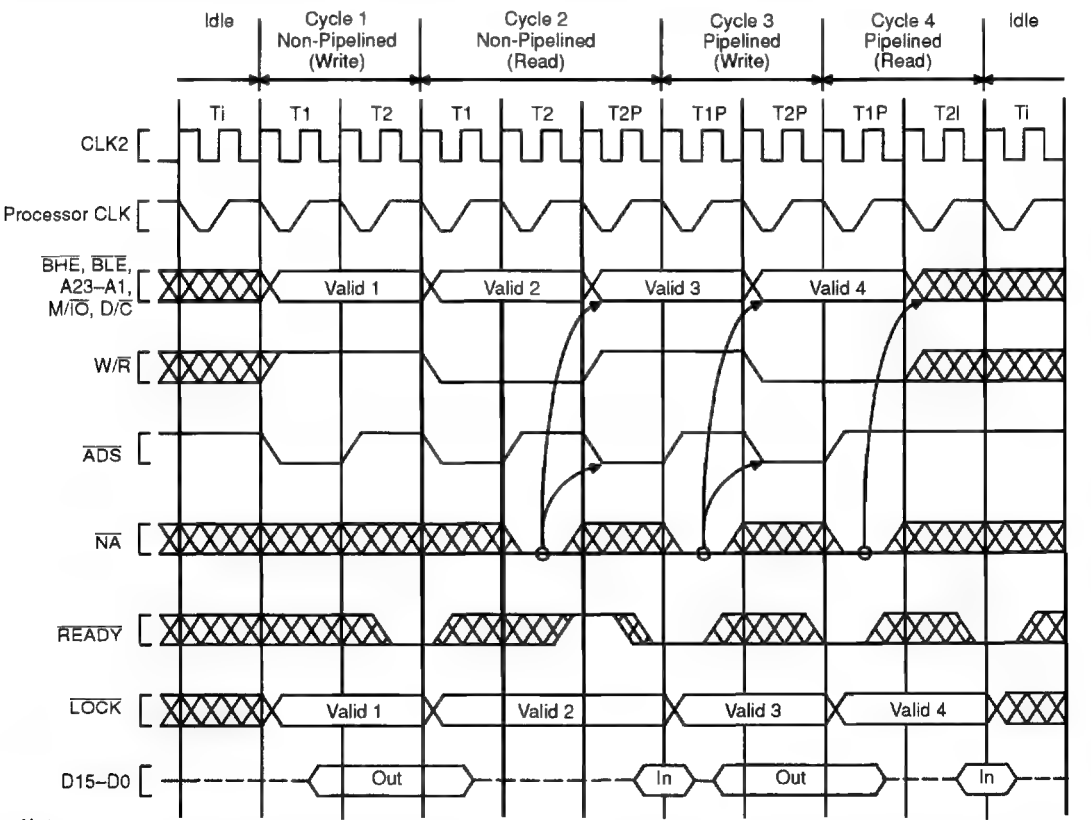
Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and $\overline{NA}$ is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle, if a bus request is pending internally, or Ti, if there is no bus request pending, or Th, if the HOLD input is being asserted.

Use of pipelined address allows the Am386SE microprocessor to enter three additional bus states not shown in Figure 22. Figure 26, is the complete bus state diagram, including pipelined address cycles.

Pipelined Address

Address pipelining is the option of requesting the address and the bus cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with $\overline{READY}$ asserted. $\overline{ADS}$ is asserted by the Am386SE microprocessor when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the $\overline{NA}$ input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles, $\overline{NA}$ is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 23, during which $\overline{NA}$ is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).



Note:
Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, $\overline{NA}$ is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

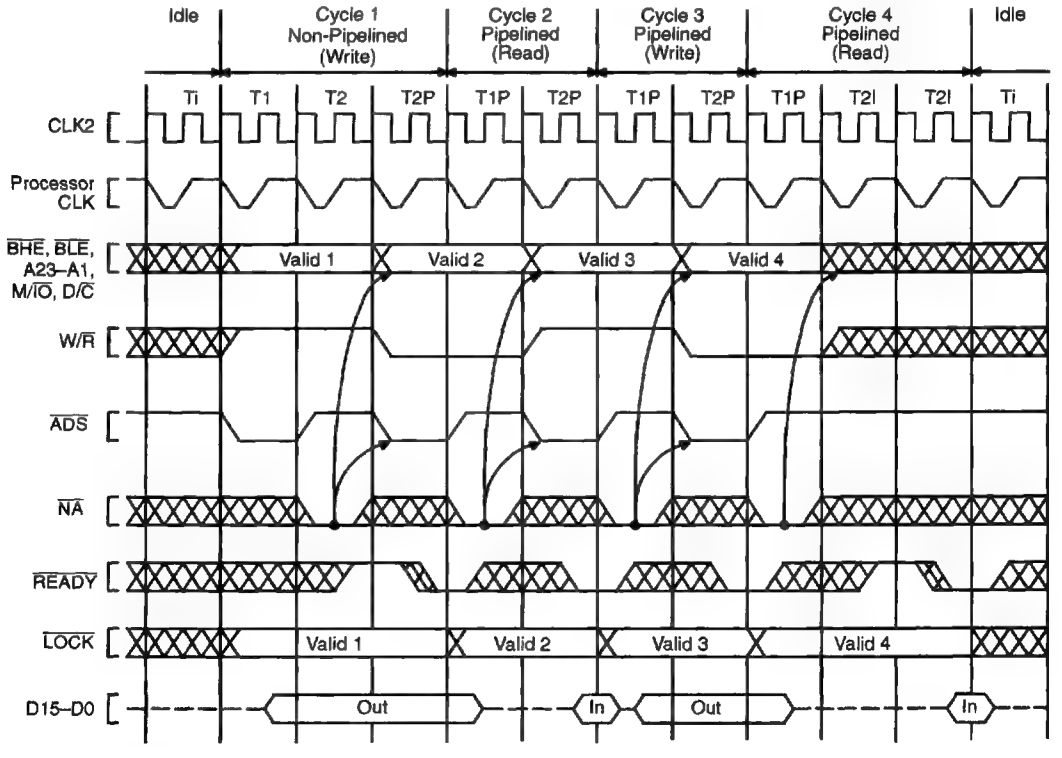
18420A–025

Figure 23. Transitioning to Pipelined Address During Burst of Bus Cycles

If $\overline{NA}$ is sampled active, the Am386SE microprocessor is free to drive the address and bus cycle definition of the next bus cycle, and assert $\overline{ADS}$, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the Am386SE CPU has the following characteristics:

1. The next address may appear as early as the bus state after $\overline{NA}$ was sampled active (see Figure 23 and Figure 24). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address will not be available immediately after $\overline{NA}$ is asserted and T2I is entered instead of T2P (see Figure 25, Cycle 3). Provided the current bus cycle is not yet acknowledged by $\overline{READY}$ asserted, T2P will be entered as soon as the Am386SE microprocessor does drive the next address. External hardware should therefore observe the $\overline{ADS}$ output as confirmation the next address is actually being driven on the bus.
2. Any address which is validated by a pulse on the $\overline{ADS}$ output will remain stable on the address pins for at least two processor clock periods. The Am386SE CPU cannot produce a new address more frequently than every two processor clock periods (see Figure 23, Figure 24, and Figure 25).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 25, Cycle 1).



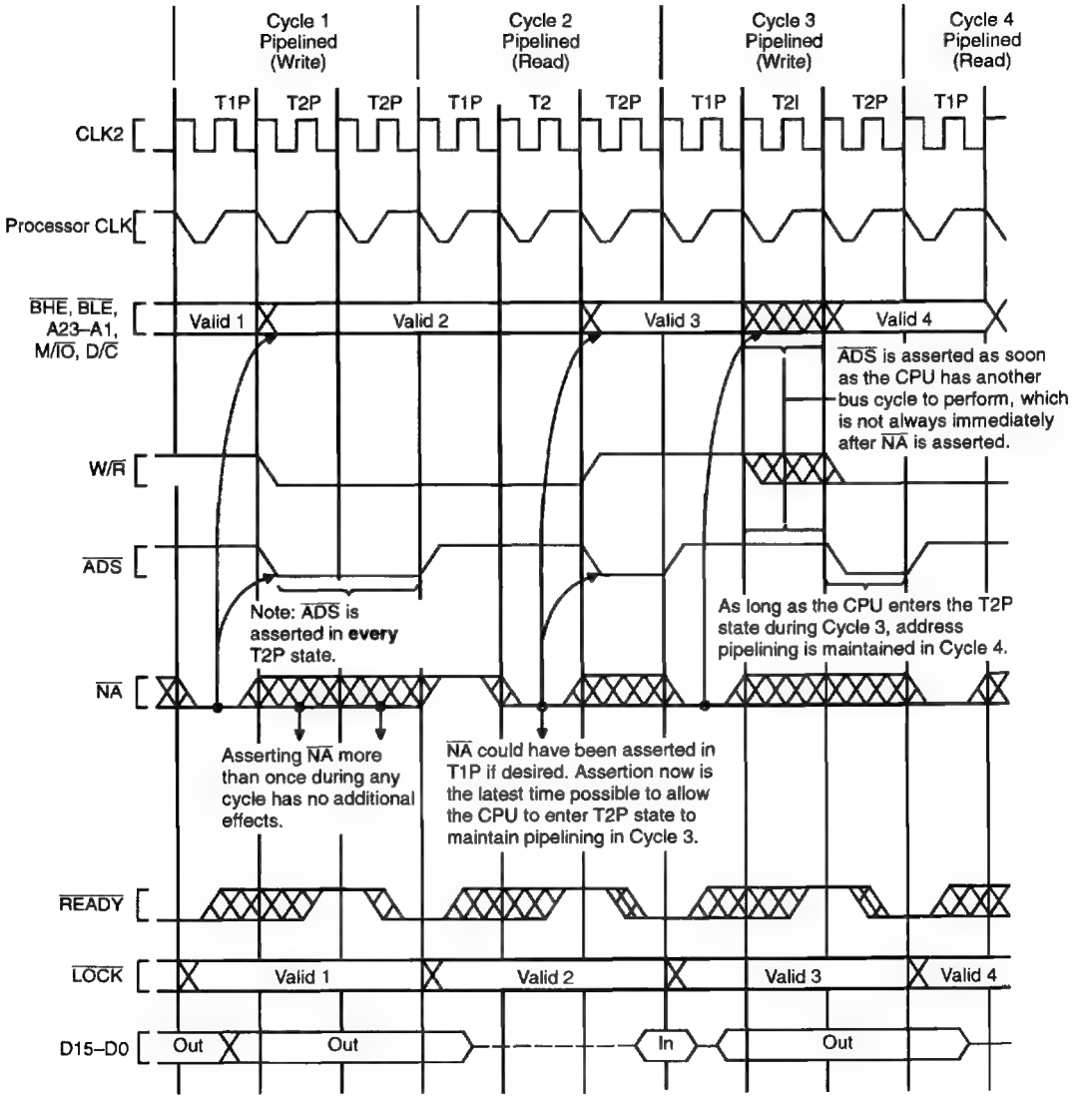
18420A–026

Note:
Following any idle bus state (Ti) the address is always non-pipelined and $\overline{NA}$ is only sampled during wait states. To start address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (Cycle 1 above). The pipelined cycles (2, 3, and 4 above) are shown with various numbers of wait states.

Figure 24. Fastest Transition to Pipelined Address Following Idle Bus State

The complete bus state transition diagram, including operation with pipelined address, is given in Figure 33. Note that it is a superset of the diagram for non-pipelined address only, and the three additional bus states for pipelined address are drawn in bold.

The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.



18420A–027

Figure 25. Details of Address Pipelining During Cycles with Wait States

Initiating and Maintaining Pipelined Address

Using the state diagram Figure 26, observe the transitions from an idle state (Ti) to the beginning of a pipelined bus cycle (T1P). From an idle state (Ti) the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided $\overline{NA}$ is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in bold below:

Ti, Ti, Ti, T1–T2–T2P, T1P–T2P,

Idle States	Non-Pipelined Cycle	Pipelined Cycle

T1-T2-T2P are the states of the bus cycle that establish address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

Th, Th, Th, T1–T2–T2P, T1P–T2P,

Hold Acknowledge States	Non-Pipelined Cycle	Pipelined Cycle

The transition to pipelined address is shown functionally by Figure 24, Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3, and 4, which are pipelined. The $\overline{NA}$ input is asserted at the appropriate time to select address pipelining for Cycle 2, 3, and 4.

Once a bus cycle is in progress and the current address has been valid for one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged. Sampling begins in T2 during Cycle 1 in Figure 24. Once $\overline{NA}$ is sampled active during the current cycle, the Am386SE microprocessor is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 24, Cycle 1 for example, the next address is driven during state T2P. Thus, Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as $\overline{READY}$ asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 24, Cycle 1 and Figure 23, Cycle 2. Figure 24 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 23, Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 ($\overline{NA}$ is asserted at that time), and T2P (provided the Am386SE microprocessor has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2, and T2P) are required in a bus cycle performing a transition from non-pipelined address into pipelined address timing (e.g., Figure 24, Cycle 1). Figure 24, Cycles 2, 3, and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting $\overline{NA}$ and detecting that the Am386SE microprocessor enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of $\overline{ADS}$. Figure 23 and Figure 24, however, each show pipelining ending after Cycle 4, because Cycle 4 ends in T2I. This indicates the Am386SE CPU did not have an internal bus request prior to the acknowledgment of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

Realistically, address pipelining is almost always maintained as long as $\overline{NA}$ is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore, address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD inactive), and $\overline{NA}$ is sampled active in each of the bus cycles.

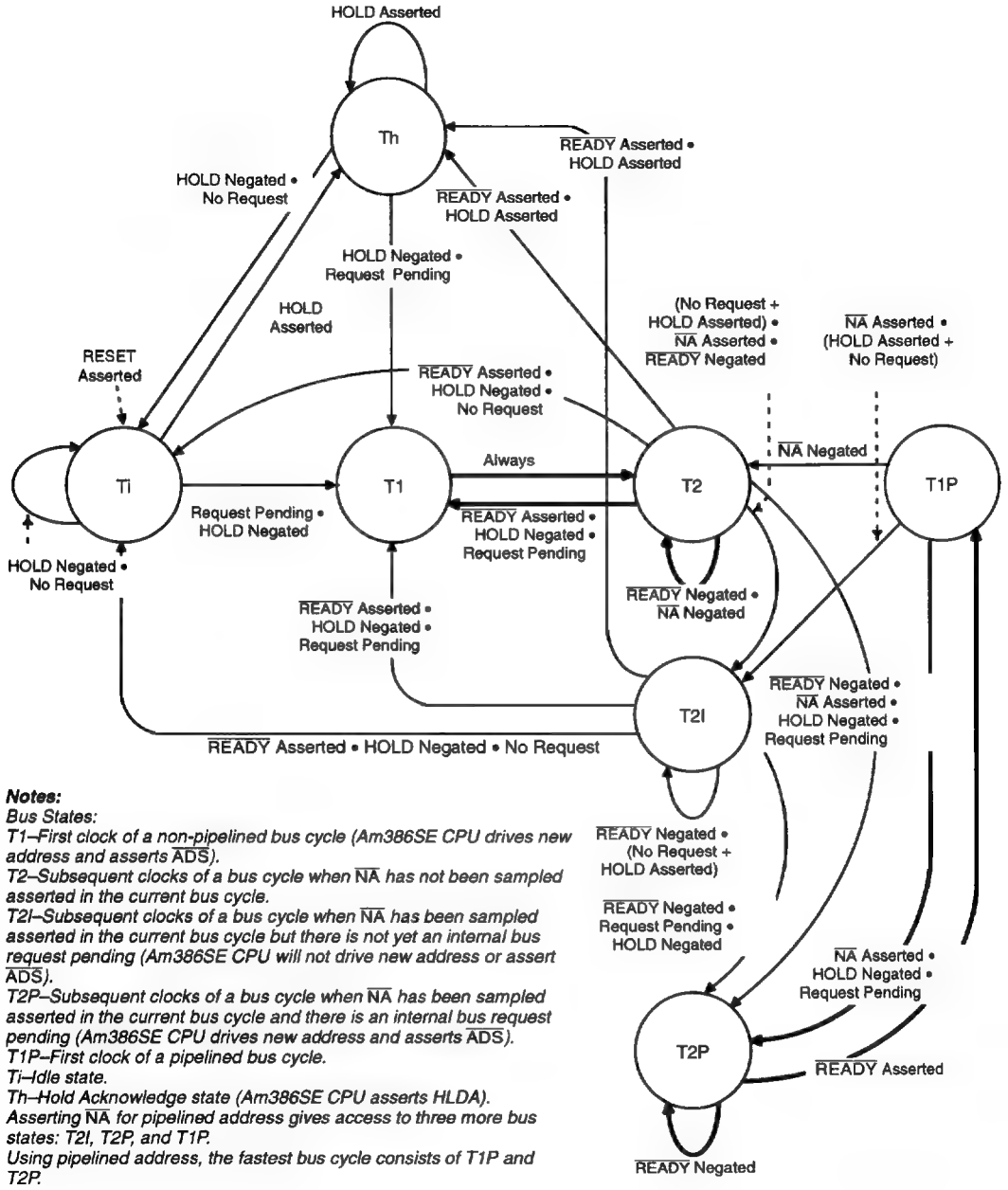
Interrupt Acknowledge (INTA) Cycles

In response to an interrupt request on the INTR input when interrupts are enabled, the Am386SE microprocessor performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by $\overline{READY}$ sampled active (see Figure 27).

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A23–A3, A1, $\overline{BLE}$ Low, A2 and $\overline{BHE}$ High). The byte address driven during the second interrupt acknowledge cycle is 0 (A23–A1, $\overline{BLE}$ Low, and $\overline{BHE}$ High).

The $\overline{LOCK}$ output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states (Ti) are inserted by the Am386SE microprocessor between the two interrupt acknowledge cycles for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D15–D0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the Am386SE microprocessor will read an external interrupt vector from D7–D0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.



Notes:
Bus States:
T1–First clock of a non-pipelined bus cycle (Am386SE CPU drives new address and asserts $\overline{ADS}$).
T2–Subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
T2I–Subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (Am386SE CPU will not drive new address or assert $\overline{ADS}$).
T2P–Subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle and there is an internal bus request pending (Am386SE CPU drives new address and asserts $\overline{ADS}$).
T1P–First clock of a pipelined bus cycle.
Ti–Idle state.
Th–Hold Acknowledge state (Am386SE CPU asserts HLDA).
Asserting $\overline{NA}$ for pipelined address gives access to three more bus states: T2I, T2P, and T1P.
Using pipelined address, the fastest bus cycle consists of T1P and T2P.

Figure 26. Complete Bus States (Including Pipelined Address) 18420A–028

Halt Indication Cycle

The execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown in Figure 28, Bus Cycle Definition Signals, and an address of 2. The halt indication cycle must be acknowledged by $\overline{\text{READY}}$ asserted. A halted Am386SE CPU resumes execution when INTR (if interrupts are enabled), NMI, or RESET is asserted.

Shutdown Indication Cycle

The Am386SE microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in the Bus Cycle Definition Signals section and an address of 0. The shutdown indication cycle must be acknowledged by $\overline{\text{READY}}$ asserted. A shut-down Am386SE microprocessor resumes execution when NMI or RESET is asserted (see Figure 29).

Entering and Exiting Hold Acknowledge

The Bus Hold Acknowledge state (Th) is entered in response to the HOLD input being asserted. In the Bus Hold Acknowledge state, the Am386SE microprocessor floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the Am386SE CPU remains in the Bus Hold Acknowledge state. In the Bus Hold Acknowledge state, all inputs except HOLD, $\overline{\text{FLT}}$, and RESET are ignored.

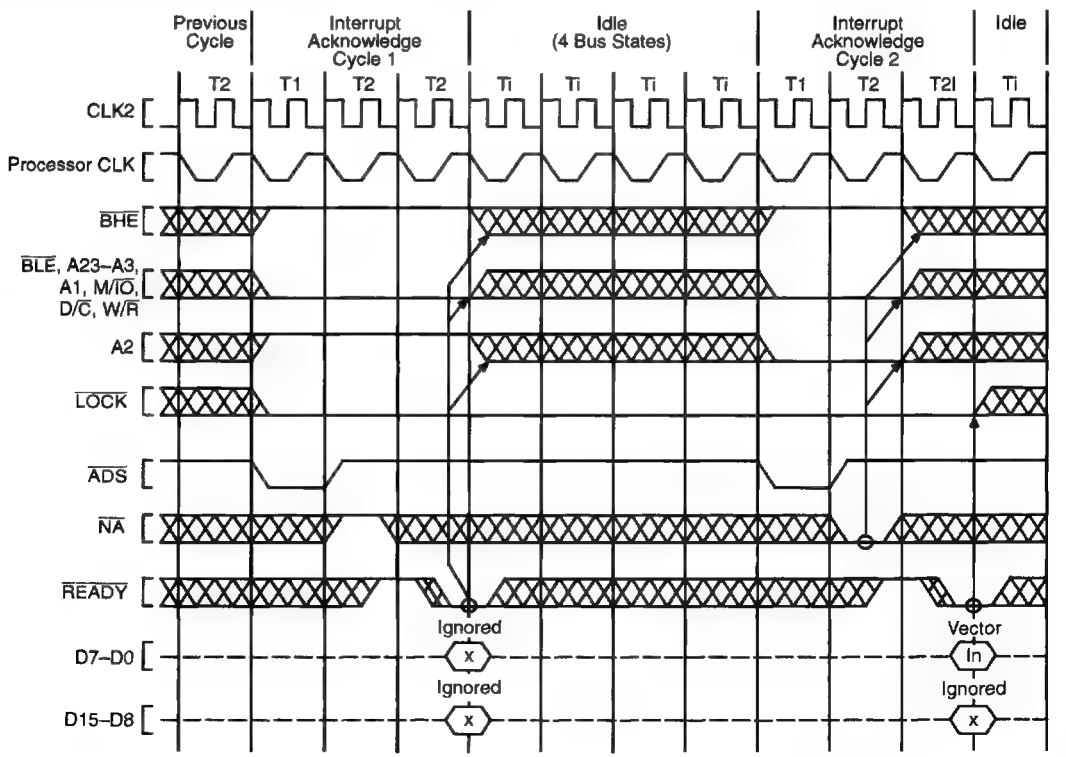


Figure 27. Interrupt Acknowledge Cycles 18420A–029

Note:
Interrupt Vector (0–255) is read on D7–D0 at end of second Interrupt Acknowledge bus cycle. Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting $\overline{\text{NA}}$ has no practical effect. Choose the approach which is simplest for your system hardware design.

Th may be entered from a bus idle state, as in Figure 30, or after the acknowledgment of the current physical bus cycle, if the $\overline{LOCK}$ signal is not asserted, as in Figure 31 and Figure 32.

Th is exited in response to the HOLD input being negated. The following state will be Ti if no bus request is pending, as in Figure 30. The following bus state will be T1 if a bus request is internally pending, as in Figure 31 and Figure 32. Th is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI in- put while in Th, the event is remembered as a non-mask- able interrupt 2 and is serviced when Th is exited, unless the Am386SE microprocessor is reset before Th is exited.

Reset During Hold Acknowledge

RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the Am386SE microprocessor drives its pins to defined states during reset, as in Table 14 (Pin State During Reset on page 39), and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the Am386SE CPU enters the Hold Acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the Am386SE microprocessor would otherwise perform its first bus cycle.

FLOAT

Activating the $\overline{FLT}$ input floats all Am386SE microprocessor bidirectional and output signals, including HLDA. Asserting $\overline{FLT}$ isolates the Am386SE microprocessor from the surrounding circuitry.

As the Am386SE microprocessor is packaged in a surface mount PQFP, it cannot be removed from the motherboard when In-Circuit Emulation (ICE) is needed. The $\overline{FLT}$ input allows the Am386SE CPU to be electrically isolated from the surrounding circuitry. This allows connection of an emulator to the Am386SE microprocessor PQFP without removing it from the PCB. This method of emulation is referred to as ON-Circuit Emulation (ONCE).

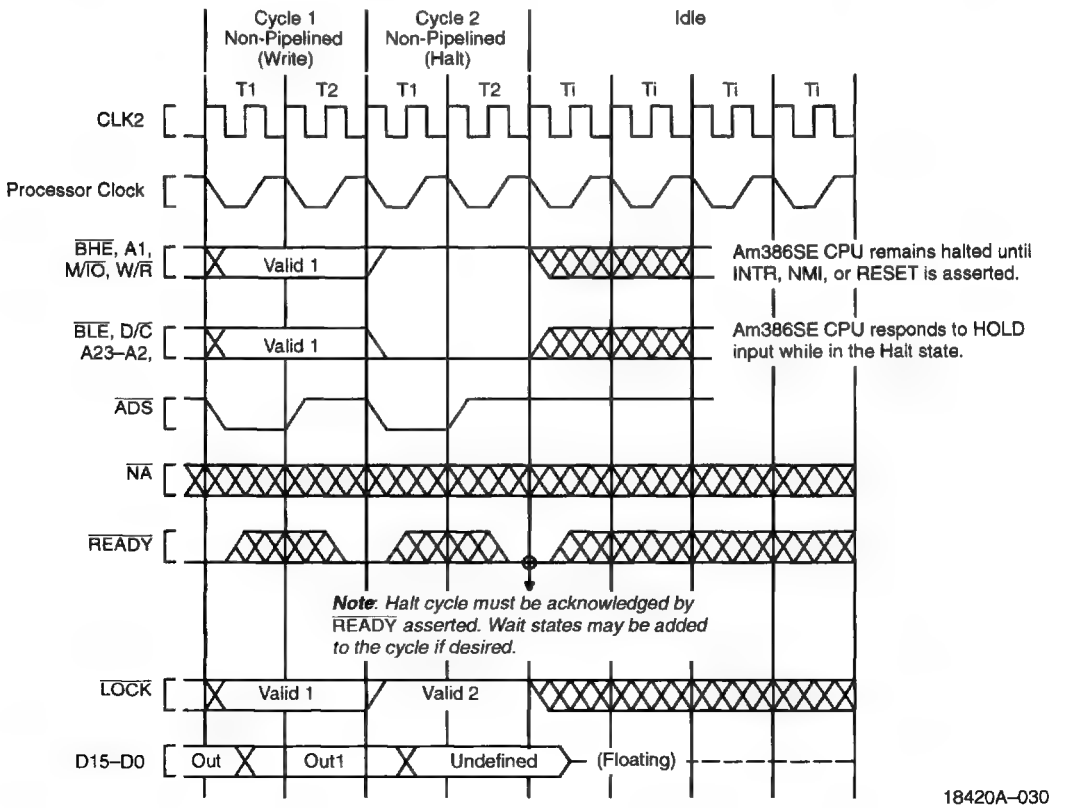


Figure 28. Example Halt Indication Cycle from Non-Pipelined Cycle

Entering and Exiting FLOAT

$\overline{FLT}$ is an asynchronous, active Low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the Am386SE microprocessor (Figure 34). $\overline{FLT}$ must be held Low for a minimum of 16 CLK2 cycles. RESET should be asserted and held asserted until after $\overline{FLT}$ is deasserted. This will ensure that the Am386SE CPU will exit FLOAT in a valid state.

Asserting the $\overline{FLT}$ input unconditionally aborts the current bus cycle and forces the Am386SE CPU into the FLOAT mode. Since activating $\overline{FLT}$ unconditionally forces the Am386SE CPU into FLOAT mode, the Am386SE microprocessor is not guaranteed to enter FLOAT in a valid state. After deactivating $\overline{FLT}$, the Am386SE CPU is not guaranteed to exit FLOAT mode in a valid state. This is not a problem, as the $\overline{FLT}$ pin is meant to be used only during ONCE. After exiting FLOAT, the Am386SE microprocessor must be reset to return it to a valid state. Reset should be asserted before $\overline{FLT}$ is deasserted. This will ensure that the Am386SE CPU will exit FLOAT in a valid state.

$\overline{FLT}$ has an internal pull-up resistor, and if it is not used it should be unconnected.

Bus Activity During and Following Reset

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states and Bus Hold Acknowledge states discontinued, so that the reset state is established.

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the Am386SE microprocessor, and at least 80 CLK2 periods if self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

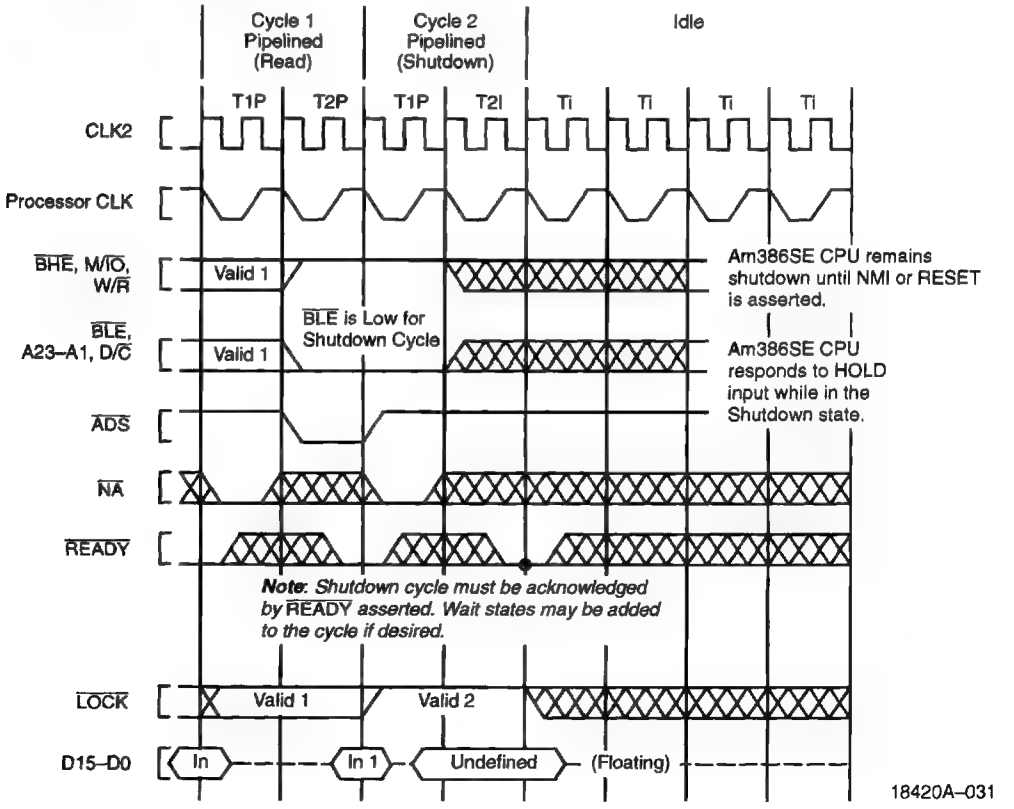


Figure 29. Example Shutdown Indication Cycle from Non-Pipelined Cycle

Provided the RESET falling edge meets setup and hold times (t25 and t26), the internal processor clock phase is defined at that time as illustrated by Figure 33 and Figure 40.

A self-test may be requested at the time RESET goes inactive by having the $\overline{BUSY}$ input at a Low level, as shown in Figure 33. The self-test requires approximately (2^{20} + 60) CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the Am386SE microprocessor attempts to proceed with the reset sequence afterwards.

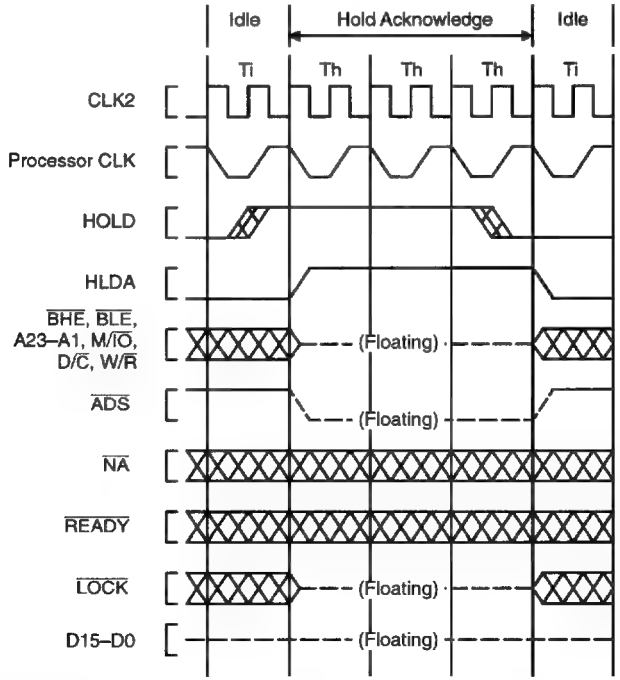
After the RESET falling edge (and after the self-test if it was requested), the Am386SE microprocessor performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

Self-Test Signature

Upon completion of self-test (if self-test was requested by driving $\overline{BUSY}$ Low at the falling edge of RESET) the EAX register will contain a signature of 00000000H, indicating the Am386SE microprocessor passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all revision levels. Any non-zero signature indicates the unit is faulty.

Component and Revision Identifiers

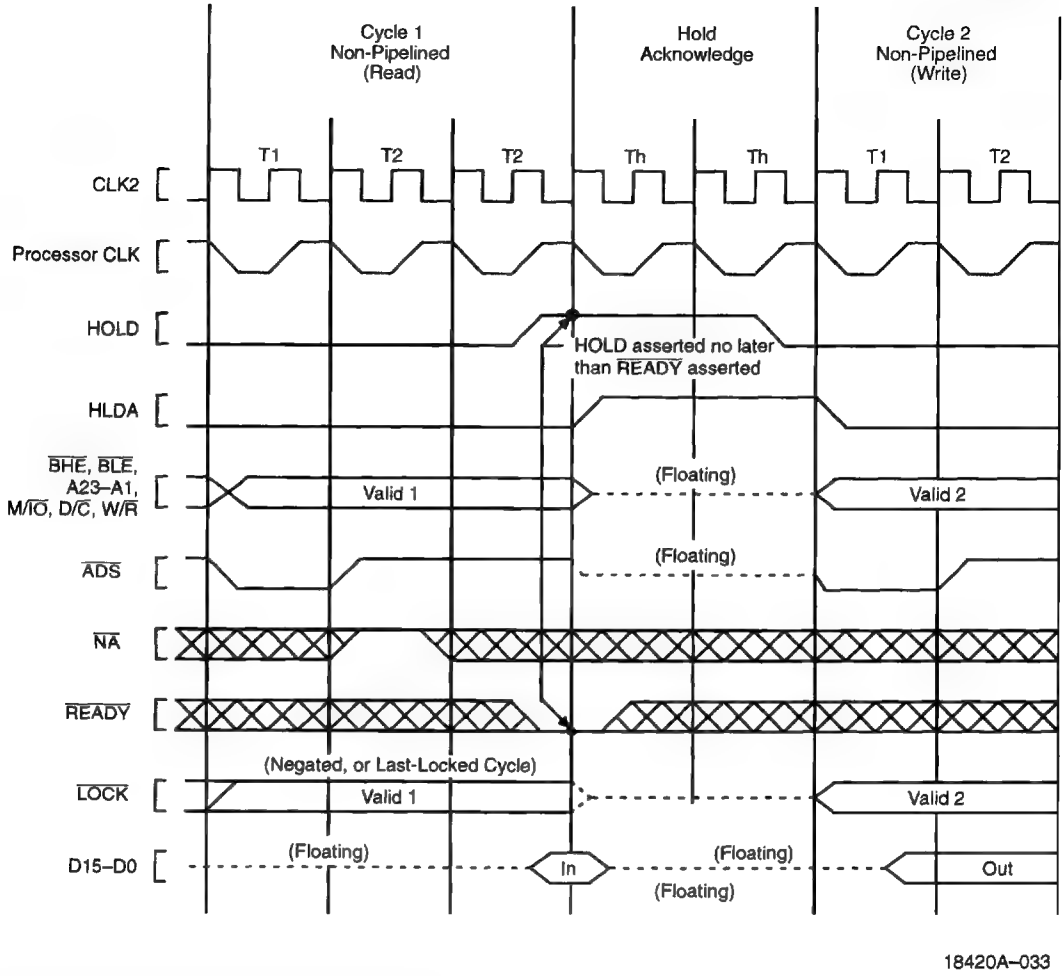
To assist users the Am386SE microprocessor, after reset, holds a component identifier and revision identifier in its DX register. The Am386SE microprocessor, after RESET, holds a component identifier and revision identifier in its DX register. In the DH portion, the lower nibble 3H refers to the Am386 CPU architecture. The upper nibble 2H refers to the second version of the Am386 microprocessor family. In the lower DL portion, the upper nibble EH identifies the processor as a member of the Am386 E Series. The lower nibble holds the unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The Am386SE microprocessor revision identifier will track that of the Am386DE CPU where possible.



18420A–032

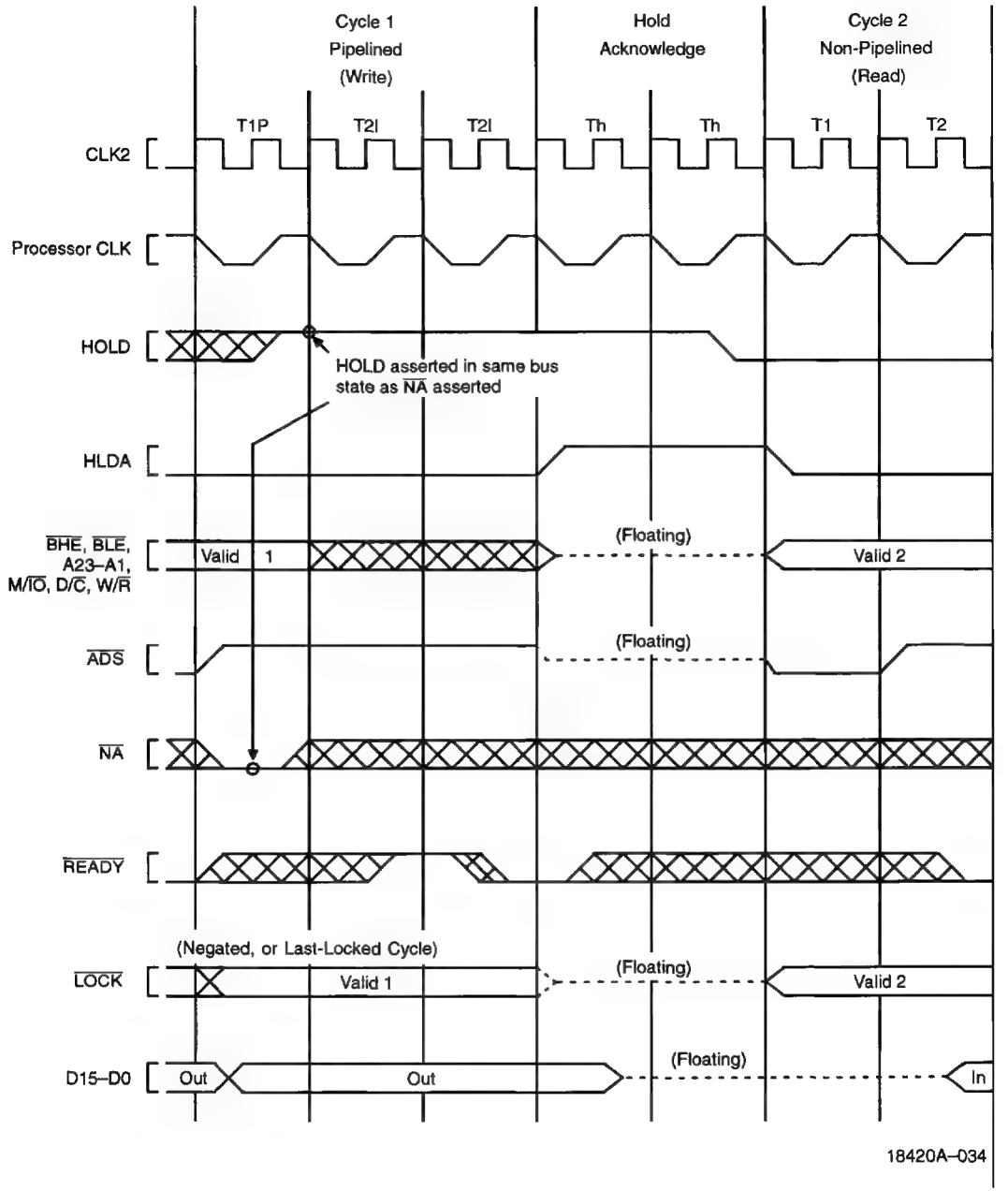
Note:
For maximum design flexibility, the Am386SE CPU has no internal pull-up resistors on its outputs. The design may require an external pull-up on $\overline{ADS}$ and other outputs to keep them negated during float periods.

Figure 30. Requesting Hold from Idle Bus



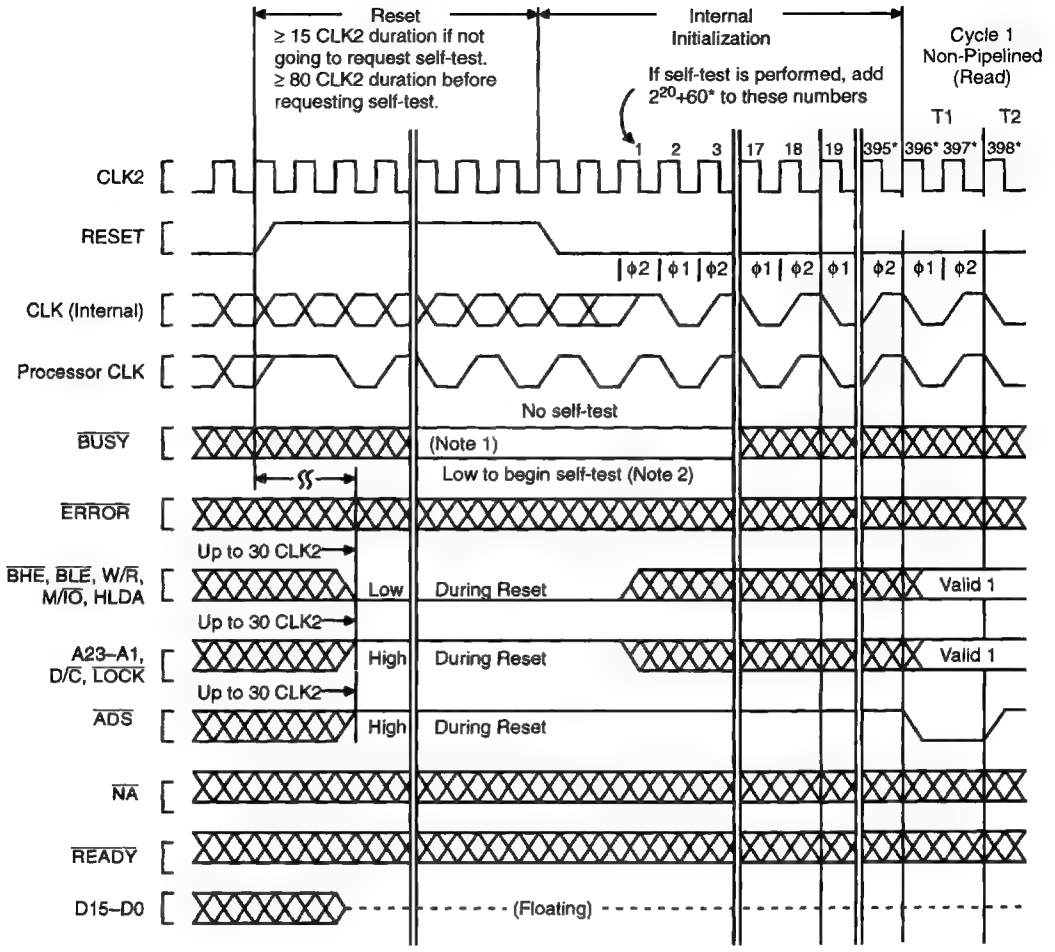
Note:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t23 and t24) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 31. Requesting Hold from Active Bus ($\overline{NA}$ Inactive)



Note:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t23 and t24) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

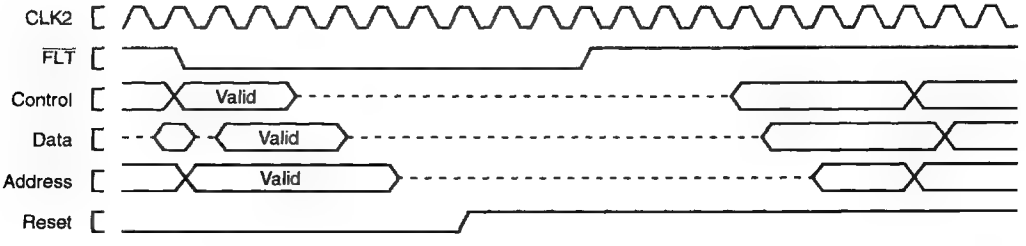
Figure 32. Requesting Hold from Idle Bus (NA Active)



Notes:
**Approximately*
1. BUSY should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested, the outputs remain in their reset state as shown here.

18420A–035

Figure 33. Bus Activity from Reset Until First Code Fetch



18420A–036

Figure 34. Entering and Exiting FLT

The revision identifier is intended to assist users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed.

Coprocessor Interfacing

The Am386SE microprocessor provides an automatic interface for a 387SX math coprocessor. A 387SX math coprocessor uses an I/O mapped interface driven automatically by the Am386SE CPU and assisted by three dedicated signals: $\overline{BUSY}$, $\overline{ERROR}$, and PEREQ.

As the Am386SE microprocessor begins supporting a math coprocessor instruction, it tests the $\overline{BUSY}$ and $\overline{ERROR}$ signals to determine if the coprocessor can accept its next instruction. Thus, the $\overline{BUSY}$ and $\overline{ERROR}$ inputs eliminate the need for any preamble bus cycles for communication between processor and math coprocessor. A 387SX math coprocessor can be given its command op-code immediately. The dedicated signals provide instruction synchronization and eliminate the need of using the WAIT op-code (9BH) for 387SX math coprocessor instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 math coprocessor).

Custom coprocessors can be included in Am386SE microprocessor-based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol primitives. Instead, memory-mapped or I/O-mapped interfaces may use all applicable instructions for high-speed coprocessor communication. The $\overline{BUSY}$ and $\overline{ERROR}$ inputs of the Am386SE microprocessor may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the WAIT op-code (9BH). The WAIT instruction will wait until the $\overline{BUSY}$ input is inactive (interruptible by an NMI or enabled INTR input), but generates an Exception 16 fault if the $\overline{ERROR}$ pin is active when the $\overline{BUSY}$ goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the Am386SE CPU's on-chip segmentation mechanism. If the custom interface is I/O-mapped, protection of the interface can be provided with the IOPL (I/O Privilege Level) mechanism.

A 387SX math coprocessor interface is I/O mapped as shown in Table 16. Note that 387SX math coprocessor interface addresses are beyond the 0H–0FFFFH range for programmed I/O. When the Am386SE microprocessor supports the 387SX math coprocessor, the Am386SE CPU automatically generates bus cycles to the coprocessor interface addresses.

Table 16. Math Coprocessor Port Address

Address in Am386SE CPU I/O Space	387SX-Compatible Math Coprocessor Register
8000F8H	Op-Code Register
8000CH/8000FEH*	Operand Register

Note: **Generated as 2nd bus cycle during Dword transfer.*

Table 17. Connections for $\overline{CMD0}$ and $\overline{CMD1}$ Inputs for a 387SX

Signal	Connection
$\overline{CMD0}$	Connected directly to Am386SE CPU A2 signal
$\overline{CMD1}$	Connected to ground

To correctly map 387SX math coprocessor registers to the appropriate I/O addresses, connect the $\overline{CMD0}$ and $\overline{CMD1}$ lines of a 387SX math coprocessor, as listed in Table 17.

Software Testing for Math Coprocessor Presence

When software is used to test for math coprocessor (387SX) presence, it should use only the following math coprocessor op-codes: FINIT, FNINIT, FSTCW mem, FSTSW mem, and FSTSW AX. To use other math coprocessor op-codes when a math coprocessor is known to be not present, first set EM = 1 in the Am386SE CPU's CR0 register.

PACKAGE THERMAL SPECIFICATIONS

The Am386SE microprocessor is specified for operation at a case temperature. The case temperature may be measured in any environment to determine whether the Am386SE CPU is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as Tc is not violated. The ambient temperature can be calculated from the θjc and θja from the following equations:

$$Tj = Tc + P \bullet \theta jc$$
$$Ta = Tj - P \bullet \theta ja$$
$$Tc = Ta + P \bullet [\theta ja - \theta jc]$$

ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the Am386SE microprocessor, and its electrical specifications.

Power and Grounding

The Am386SE CPU has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 V_{CC} and 18 V_{SS} pins separately feed functional units of the Am386SE microprocessor.

Power and ground connections must be made to all external V_{CC} and V_{SS} pins of the Am386SE microprocessor. On the circuit board, all V_{CC} pins should be connected on a V_{CC} plane, and V_{SS} pins should be connected on a GND plane.

Power Decoupling Recommendations

Liberal decoupling capacitors should be placed near the Am386SE microprocessor. The Am386SE CPU driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Am386SE microprocessor and decoupling capacitors as much as possible.

Resistor Recommendations

The $\overline{ERROR}$, $\overline{FLT}$, and $\overline{BUSY}$ inputs have internal pull-up resistors of approximately 20 Kohms, and the PEREQ input has an internal pull-down resistor of approximately 20 Kohms, built into the Am386SE microprocessor to keep these signals inactive when a 387SX-compatible math coprocessor is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 18 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. NC pins should always remain unconnected. Connection of NC pins to V_{CC} or V_{SS} will result in component malfunction or incompatibility with future steppings of the Am386SE CPU.

Particularly when not using the interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND.

Pin	Signal
40	INTR
38	NMI
4	HOLD

If not using address pipelining, connect pin 6 ($\overline{NA}$) through a pull-up in the range of 20 Kohms to V_{CC}.

Table 18. Recommended Resistor Pull-Ups to V_{CC}

Pin	Signal	Pull-Up Value	Purpose
16	$\overline{ADS}$	20 Kohms ±10%	Lightly pull $\overline{ADS}$ inactive during Am386SE CPU Hold Acknowledge states
26	$\overline{LOCK}$	20 Kohms ±10%	Lightly pull $\overline{LOCK}$ inactive during Am386SE CPU Hold Acknowledge states

ABSOLUTE MAXIMUM RATINGS

Storage Temperature ..... –65°C to +150°C
Ambient Temperature Under Bias . –65°C to +125°C

Stresses above those listed may cause permanent damage to the device. Functionality at or above these limits is not implied. Exposure to ABSOLUTE MAXIMUM RATING conditions for extended periods of time may affect device reliability.

OPERATING RANGES

Supply Voltage with respect to V_{SS} . –0.5 V to +7.0 V
Voltage on Other Pins ..... –0.5 V to V_{cc} +0.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over COMMERCIAL operating ranges (25 MHz)

V_{CC} = 3.6 V – 5.5 V; T_{CASE} = 0°C to +100°C (Extended Temperature T_{CASE} = –40°C to 100°C)

			Preliminary		
Symbol	**Parameter Description**	**Notes**	**Min**	**Max**	**Unit**
V_{IL}	Input Low Voltage	(Note 1)	–0.3	+0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.3	V
V_{ILC}	CLK2 Input Low Voltage	(Note 1)	–0.3	+0.8	V
V_{IHC}	CLK2 Input High Voltage (25 MHz)		2.7	V_{CC} + 0.3	V
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A23–A1, D15–D0 I_{OL} = 5 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA	(Note 6)		0.45 0.45	V V
V_{OH}	Output High Voltage I_{OH} = 1.0 mA: A23–A1, D15–D0 I_{OH} = 0.2 mA: A23–A1, D15–D0 I_{OH} = 0.9 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA I_{OH} = 0.18 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA	(Note 6)	2.4 V_{CC} – 0.5 2.4 V_{CC} – 0.5		V V V V
I_{LI}	Input Leakage Current (All pins except PEREQ, $\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	0 V ≤ V_{IN} ≤ V_{CC}		±15	µA
I_{IH}	Input Leakage Current (PEREQ pin)	V_{IH} = 2.4 V (Note 2)		200	µA
I_{IL}	Input Leakage Current ($\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	V_{IL} = 0.45 V (Note 3)		–400	µA
I_{LO}	Output Leakage Current	0.45 V ≤ V_{OUT} ≤ V_{CC}		±15	µA
I_{CC}	Supply Current CLK2 = 50 MHz: Oper. Freq. 25 MHz	V_{CC} Typ = 5.0 V I_{CC} Typ = 160		V_{CC} = 5.5 190	V mA
I_{CCSB}	Standby Current (Note 5)	I_{CCSB} Typ = 20 µA		150	µA
C_{IN}	Input or I/O Capacitance	F_C = 1 MHz (Note 4)		10	pF
C_{OUT}	Output Capacitance	F_C = 1 MHz (Note 4)		12	pF
C_{CLK}	CLK2 Capacitance	F_C = 1 MHz (Note 4)		20	pF

Notes:
Tested at minimum operating frequency of the part.
1. The Min value, –0.3, is not 100% tested.
2. PEREQ input has an internal pull-down resistor.
3. $\overline{BUSY}$, $\overline{ERROR}$, and $\overline{FLT}$ inputs each have an internal pull-up resistor.
4. Not 100% tested.
5. Measurement taken with inputs at rails; outputs unloaded; $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$ at V_{CC}; and PEREQ at GND.
6. Outputs are CMOS and will pull rail-to-rail if the load is not resistive.

DC CHARACTERISTICS over COMMERCIAL operating ranges (25 MHz)

V_{CC} = 3.0 V – 3.6 V; T_{CASE} = 0°C to +100°C (Extended Temperature T_{CASE} = –40°C to 100°C)

			Preliminary		
Symbol	**Parameter Description**	**Notes**	**Min**	**Max**	**Unit**
V_{IL}	Input Low Voltage	(Note 1)	– 0.3	+0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.3	V
V_{ILC}	CLK2 Input Low Voltage	(Note 1)	– 0.3	+0.8	V
V_{IHC}	CLK2 Input High Voltage (25 MHz)		2.4	V_{CC} + 0.3	V
V_{OL}	Output Low Voltage I_{OL} = 0.5 mA: A23–A1, D15–D0 I_{OL} = 0.5 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA I_{OL} = 2 mA: A23–A1, D15–D0 I_{OL} = 2.5 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA	(Note 5)		0.2 0.2 0.45 0.45	V V V V
V_{OH}	Output High Voltage I_{OH} = 0.1 mA: A23–A1, D15–D0 I_{OH} = 0.1 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA I_{OH} = 0.5 mA: A23–A1, D15–D0 I_{OH} = 0.5 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA	(Note 5) (Note 6)	V_{CC} – 0.2 V_{CC} – 0.2 V_{CC} – 0.45 V_{CC} – 0.45		V V V V
I_{LI}	Input Leakage Current (All pins except PEREQ, $\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	0 V ≤ V_{IN} ≤ V_{CC}		±10	µA
I_{IH}	Input Leakage Current (PEREQ pin)	V_{IH} = V_{CC} – 0.1 V V_{IH} = 2.4 V (Note 2)		300 200	µA µA
I_{IL}	Input Leakage Current ($\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	V_{IL} = 0.1 V V_{IL} = 0.45 V (Note 3)		– 300 – 200	µA µA
I_{LO}	Output Leakage Current	0.1 V ≤ V_{OUT} ≤ V_{CC}		±15	µA
I_{CC}	Supply Current (Note 6) CLK2 = 50 MHz: Oper. Freq. 25 MHz	V_{CC} = 3.3 V I_{CC} Typ = 95		V_{CC} = 3.6 V 115	 mA
I_{CCSB}	Standby Current (Note 6)	I_{CCSB} Typ = 10 µA		150	µA
C_{IN}	Input or I/O Capacitance	F_C = 1 MHz (Note 4)		10	pF
C_{OUT}	Output Capacitance	F_C = 1 MHz (Note 4)		12	pF
C_{CLK}	CLK2 Capacitance	F_C = 1 MHz (Note 4)		20	pF

Notes:

1. The Min value, –0.3, is not 100% tested.
2. PEREQ input has an internal pull-down resistor.
3. $\overline{BUSY}$, $\overline{ERROR}$, and $\overline{FLT}$ inputs each have an internal pull-up resistor.
4. Not 100% tested.
5. Outputs are CMOS and will pull rail-to-rail if the load is not resistive.
6. Inputs at rails (V_{CC} or V_{SS}).

DC CHARACTERISTICS over COMMERCIAL operating ranges (33 MHz)

V_{CC} = 4.5 V – 5.5 V; T_{CASE} = 0°C to +100°C

			Preliminary		
Symbol	**Parameter Description**	**Notes**	**Min**	**Max**	**Unit**
V_{IL}	Input Low Voltage	(Note 1)	–0.3	+0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.3	V
V_{ILC}	CLK2 Input Low Voltage	(Note 1)	–0.3	+0.8	V
V_{IHC}	CLK2 Input High Voltage		2.7	V_{CC} + 0.3	V
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A23–A1, D15–D0 I_{OL} = 5 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$ M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA	(Note 6)		 0.45 0.45	 V V
V_{OH}	Output High Voltage I_{OH} = 1 mA: A23–A1, D15–D0 I_{OH} = 0.9 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA I_{OH} = 0.2 mA: A23–A1, D15–D0 I_{OH} = 0.18 mA: $\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, $\overline{LOCK}$, $\overline{ADS}$, M/$\overline{IO}$, HLDA	(Note 6)	 2.4 2.4 V_{CC} – 0.5 V_{CC} – 0.5		 V V V V
I_{LI}	Input Leakage Current (All pins except PEREQ, $\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	0 V ≤ V_{IN} ≤ V_{CC}		±15	µA
I_{IH}	Input Leakage Current (PEREQ pin)	V_{IH} = 2.4 V (Note 2)		200	µA
I_{IL}	Input Leakage Current ($\overline{BUSY}$, $\overline{ERROR}$, $\overline{FLT}$)	V_{IL} = 0.45 V (Note 3)		–400	µA
I_{LO}	Output Leakage Current	0.45 V ≤ V_{OUT} ≤ V_{CC}		±15	µA
I_{CC}	Supply Current CLK2 = 66 MHz: Oper. Freq. 33 MHz	V_{CC} = 5.0 V I_{CC} Typ = 210		V_{CC} = 5.5 V 245	 mA
I_{CCSB}	Standby Current (Note 5)	I_{CCSB} Typ = 20 µA	20	150	µA
C_{IN}	Input Capacitance	F_C = 1 MHz (Note 4)		10	pF
C_{OUT}	Output or I/O Capacitance	F_C = 1 MHz (Note 4)		12	pF
C_{CLK}	CLK2 Capacitance	F_C = 1 MHz (Note 4)		20	pF

Notes:

Tested at minimum operating frequency of the part.

1. The Min value, –0.3, is not 100% tested.

2. PEREQ input has an internal pull-down resistor.

3. $\overline{BUSY}$, $\overline{ERROR}$, and $\overline{FLT}$ inputs each have an internal pull-up resistor.

4. Not 100% tested.

5. Measurement taken with inputs at rails; outputs unloaded; $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$ at V_{CC}; and PEREQ at GND.

6. Outputs are CMOS and will pull rail-to-rail if the load is not resistive.

SWITCHING CHARACTERISTICS

The switching characteristics given consist of output delays, input setup requirements, and input hold requirements. All switching characteristics are relative to the CLK2 rising edge crossing the 2.0 V level.

Switching characteristic measurement is defined by Figure 35. Inputs must be driven to the voltage levels indicated by Figure 35 when switching characteristics are measured. Output delays are specified with minimum and maximum limits measured, as shown. The minimum delay times are hold times provided to external circuitry. Input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct operation.

Outputs $\overline{ADS}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{BHE}$, $\overline{BLE}$, A23–A1, and HLDA only change at the beginning of phase one. D15–D0 (write cycles) only change at the beginning of phase two. The $\overline{READY}$, HOLD, $\overline{BUSY}$, $\overline{ERROR}$, PEREQ, $\overline{FLT}$, and D15–D0 (read cycles) inputs are sampled at the beginning of phase one. The $\overline{NA}$, INTR, and NMI inputs are sampled at the beginning of phase two.

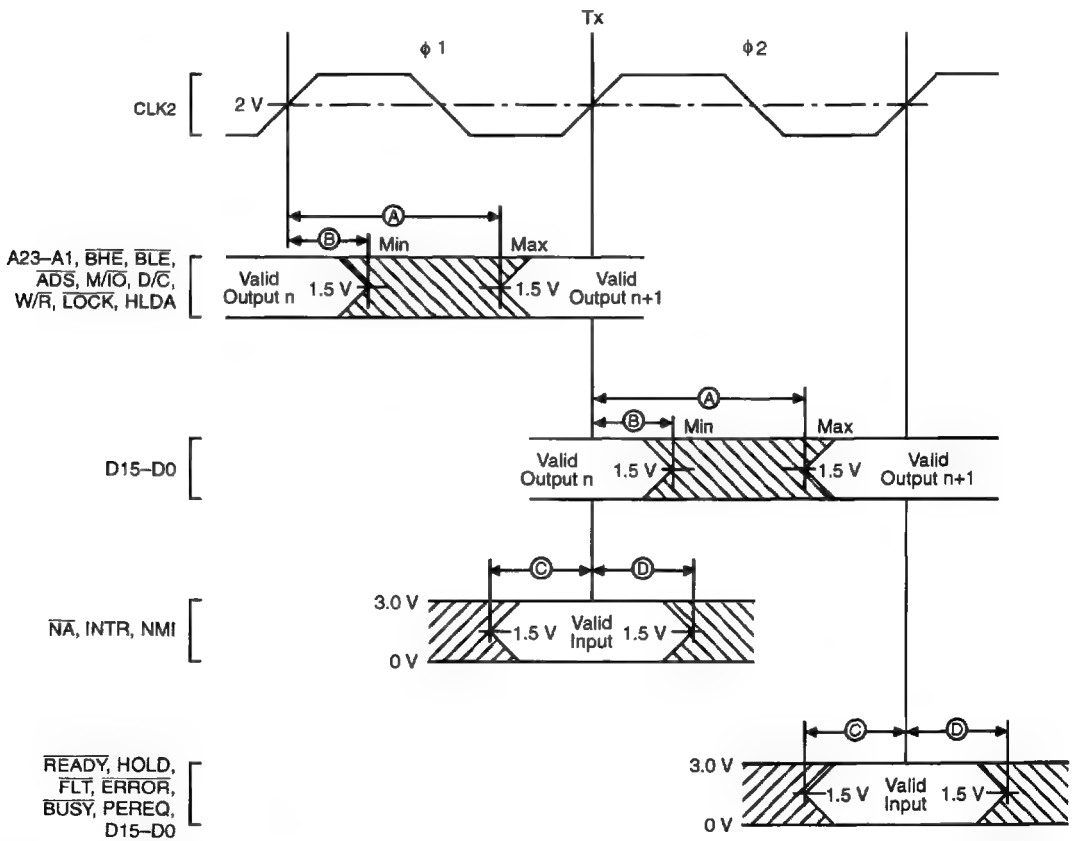


Figure 35. Drive Levels and Measurement Points for Switching Characteristics

SWITCHING CHARACTERISTICS over COMMERCIAL operating ranges (25 MHz)

V_{CC} = 3.0 – 5 V; T_{CASE} = 0°C to 100°C (Extended Temperature Range; T_{CASE} = –40°C to 100°C)

Symbol	Parameter Description	Notes	Ref. Figure	Min.	Max.	Unit
	Am386SE CPU	Half CLK2 freq.		0	25	MHz
1	Am386SE CPU		36	20		ns
2a	CLK2 High Time	at 2 V	36	7		ns
2b	CLK2 High Time	at (V_{CC} – 0.8 V)	36	4		ns
3a	CLK2 Low Time	at 2 V	36	7		ns
3b	CLK2 Low Time	at 0.8 V	36	5		ns
4	CLK2 Fall Time	(V_{CC} – 0.8 V) to 0.8 V (Note 3)	36		7	ns
5	CLK2 Rise Time	0.8 V to (V_{CC} – 0.8 V) (Note 3)	36		7	ns
6	A23–A1 Valid Delay	C_L = 50 pF	39	4	17	ns
7	A23–A1 Float Delay	(Note 1)	42	4	30	ns
8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	C_L = 50 pF	39	4	17	ns
9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	(Note 1)	42	4	30	ns
10	M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{ADS}$ Valid Delay	C_L = 50 pF	39	4	17	ns
11	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	(Note 1)	42	4	30	ns
12	D15–D0 Write Data Valid Delay	C_L = 50 pF	39,40	7	23	ns
12a	D15–D0 Write Data Hold Time	C_L = 50 pF	41	2		ns
13	D15–D0 Write Data Float Delay	(Note 1)	42	4	22	ns
14	HLDA Valid Delay	C_L = 50 pF	39	4	22	ns
14f	HLDA Float Delay		42	4	22	ns
15	$\overline{NA}$ Setup Time		38	5		ns
16	$\overline{NA}$ Hold Time		38	3		ns
19	$\overline{READY}$ Setup Time		38	9		ns
20	$\overline{READY}$ Hold Time		38	4		ns
21	D15–D0 Read Data Setup Time		38	7		ns
22	D15–D0 Read Data Hold Time		38	5		ns
23	HOLD Setup Time		38	9		ns
24	HOLD Hold Time		38	3		ns
25	RESET Setup Time		43	8		ns
26	RESET Hold Time		43	3		ns
27	NMI, INTR Setup Time	(Note 2)	38	6		ns
28	NMI, INTR Hold Time	(Note 2)	38	6		ns
29	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup TIme	(Note 2)	38	6		ns
30	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold TIme	(Note 2)	38	5		ns

Notes:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.

2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.

3. These are not tested. They are guaranteed by design characterization.

SWITCHING CHARACTERISTICS over COMMERCIAL operating ranges (33 MHz)

V_{CC} = 4.5 V – 5.5 V; T_{CASE} = 0°C to 100°C

Symbol	Parameter Description	Notes	Ref. Figure	Min.	Max.	Unit
	Am386SE CPU	Half CLK2 freq.		0	33	MHz
1	Am386SE CPU		36	15		ns
2a	CLK2 High Time	at 2 V	36	6.25		ns
2b	CLK2 High Time	at 3.7 V	36	4		ns
3a	CLK2 Low Time	at 2 V	36	6.25		ns
3b	CLK2 Low Time	at 0.8 V	36	4.5		ns
4	CLK2 Fall Time	3.7 V to 0.8 V (Note 3)	36		4	ns
5	CLK2 Rise Time	0.8 V to 3.7 V (Note 3)	36		4	ns
6	A23–A1 Valid Delay	C_L = 50 pF (Note 4)	39	4	15	ns
7	A23–A1 Float Delay	(Note 1)	42	4	20	ns
8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	C_L = 50 pF	39	4	15	ns
9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	(Note 1)	42	4	20	ns
10	M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{ADS}$ Valid Delay	C_L = 50 pF	39	4	15	ns
11	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	(Note 1)	42	4	20	ns
12	D15–D0 Write Data Valid Delay	C_L = 50 pF (Note 4)	39,40	7	23	ns
12a	D15–D0 Write Data Hold Time	C_L = 50 pF	41	2		ns
13	D15–D0 Write Data Float Delay	(Note 1)	42	4	17	ns
14	HLDA Valid Delay	C_L = 50 pF	39	4	20	ns
14f	HLDA Float Delay	(Note 1)	42	4	20	ns
15	$\overline{NA}$ Setup Time		38	5		ns
16	$\overline{NA}$ Hold Time		38	2		ns
19	$\overline{READY}$ Setup Time		38	7		ns
20	$\overline{READY}$ Hold Time		38	4		ns
21	D15–D0 Read Data Setup Time		38	5		ns
22	D15–D0 Read Data Hold Time		38	3		ns
23	HOLD Setup Time		38	9		ns
24	HOLD Hold Time		38	2		ns
25	RESET Setup Time		43	5		ns
26	RESET Hold Time		43	2		ns
27	NMI, INTR Setup Time	(Note 2)	38	5		ns
28	NMI, INTR Hold Time	(Note 2)	38	5		ns
29	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup Time	(Note 2)	38	5		ns
30	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold Time	(Note 2)	38	4		ns

Notes:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.

2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.

3. These are not tested. They are guaranteed by design characterization.

4. Tested with C_L set at 50 pF and derated to support the indicated distribution capacitive load.

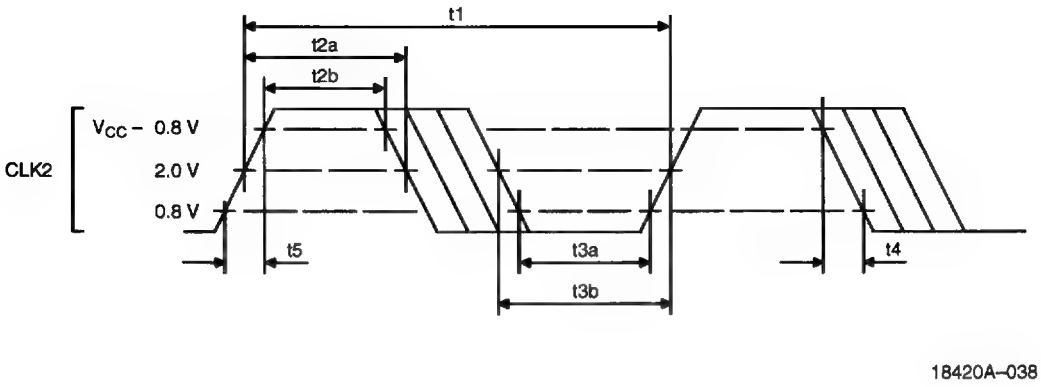


Figure 36. CLK2 Timing (25, 33 MHz)

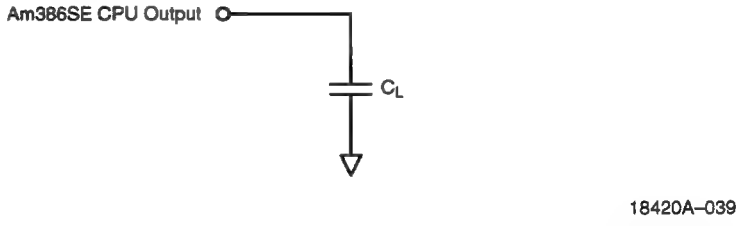


Figure 37. AC Test Circuit

SWITCHING WAVEFORMS

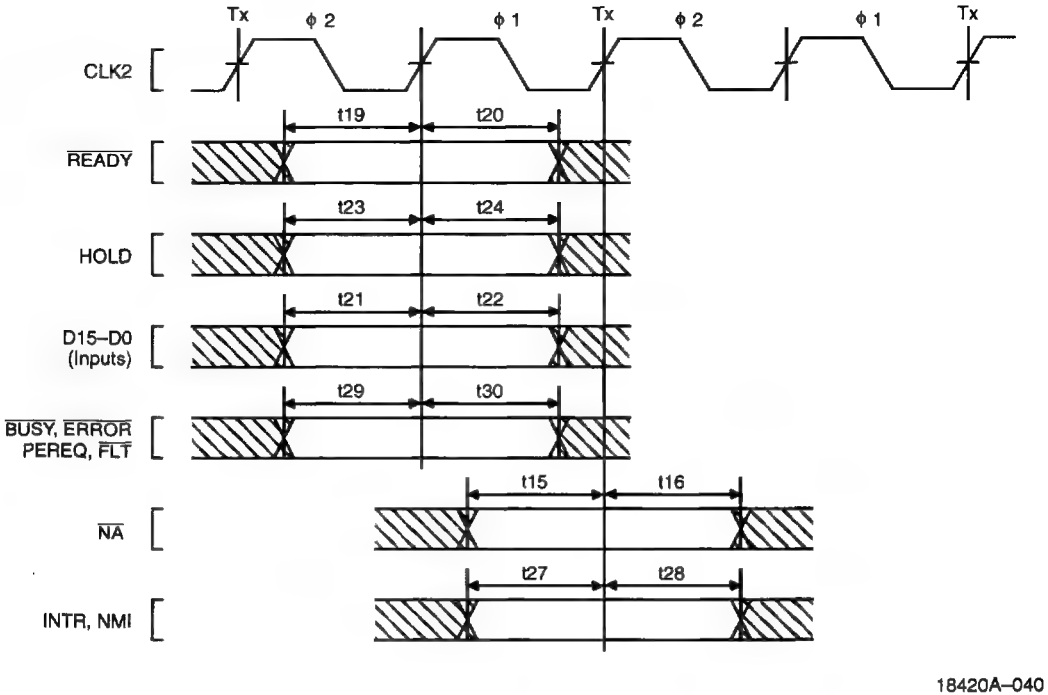


Figure 38. Input Setup and Hold Timing

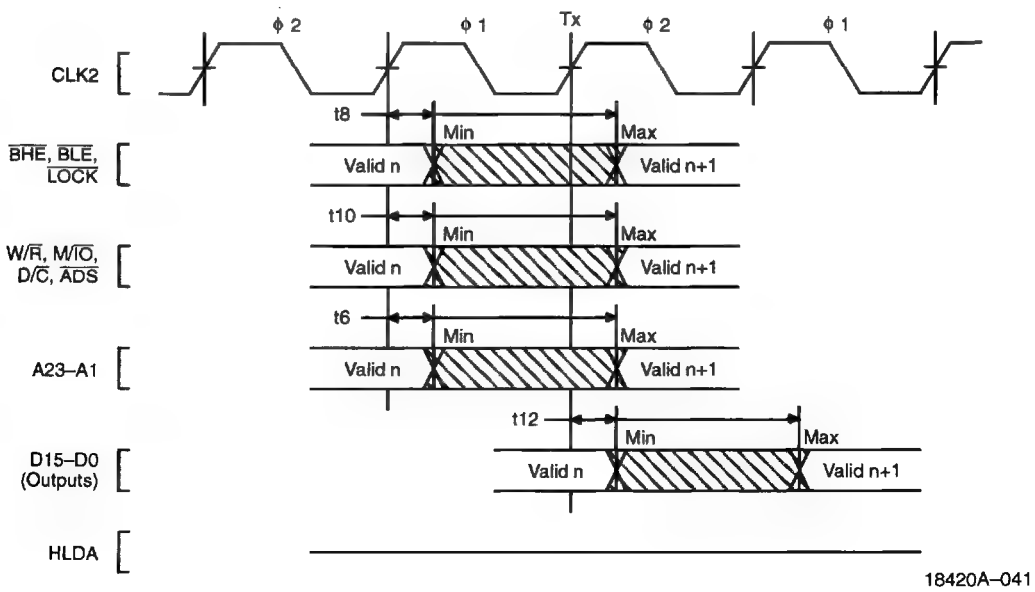
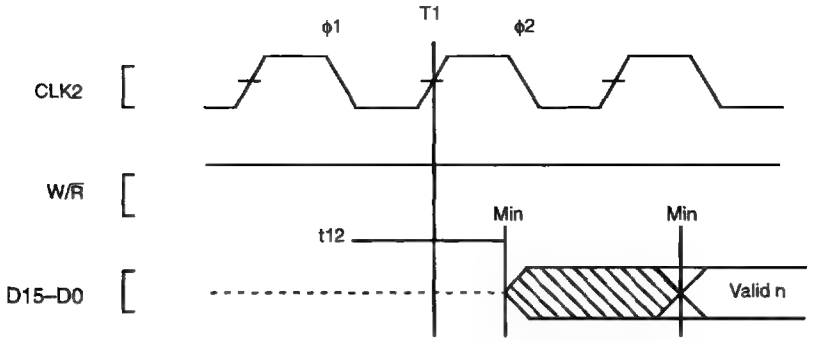


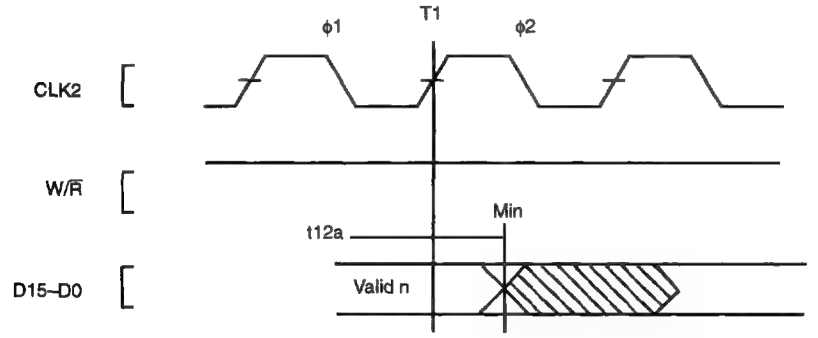
Figure 39. Output Valid Delay Timing

SWITCHING WAVEFORMS (continued)



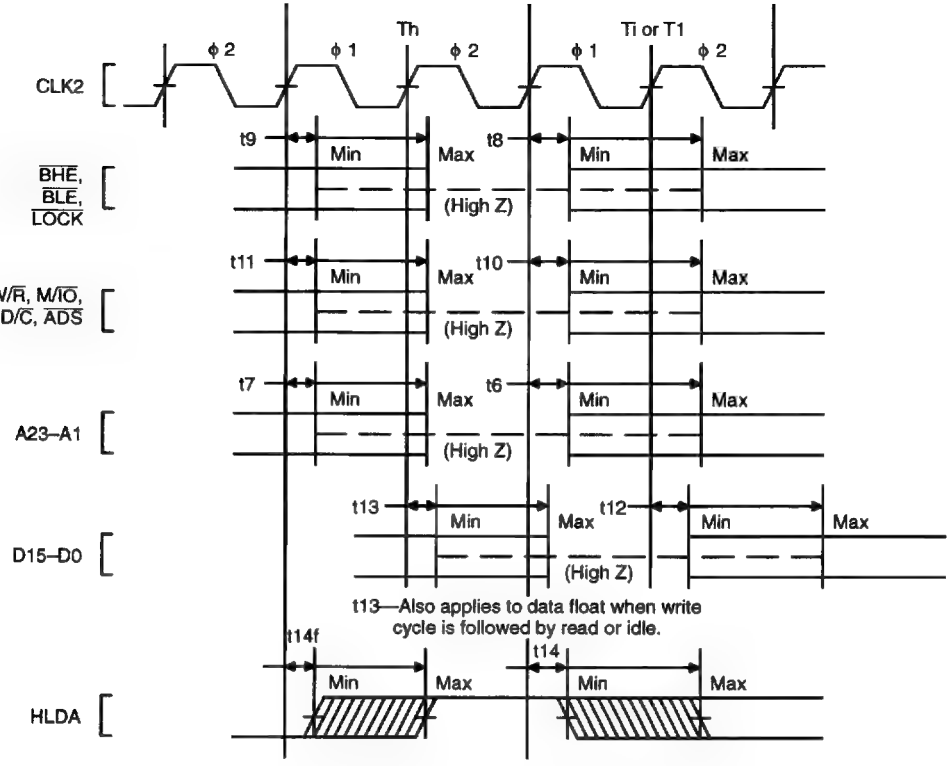
18420A–042

Figure 40. Write Data Valid Delay Timing (25 MHz)



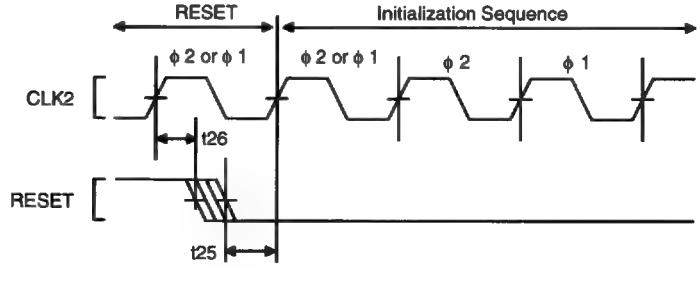
18420A–043

Figure 41. Write Data Hold Timing



18420A–044

Figure 42. Output Float Delay and HLDA Valid Delay Timing



18420A–045

Figure 43. RESET Setup and Hold Timing and Internal Phase

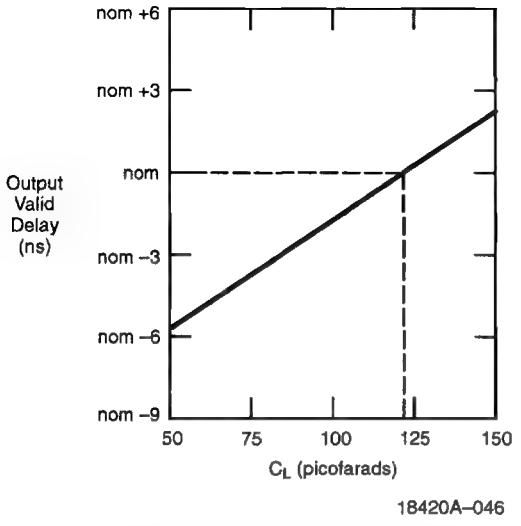


Figure 44. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L=120 pF)

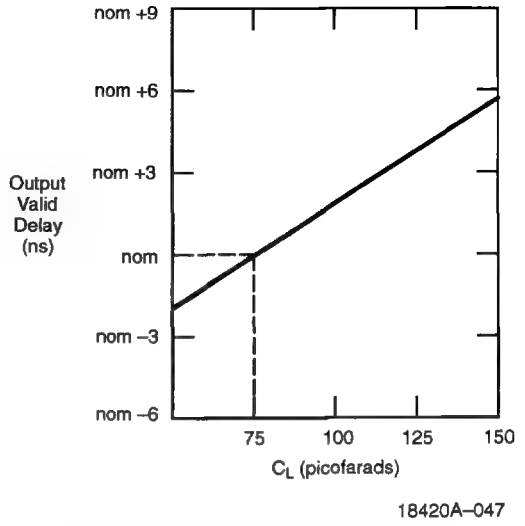


Figure 45. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L=75 pF)

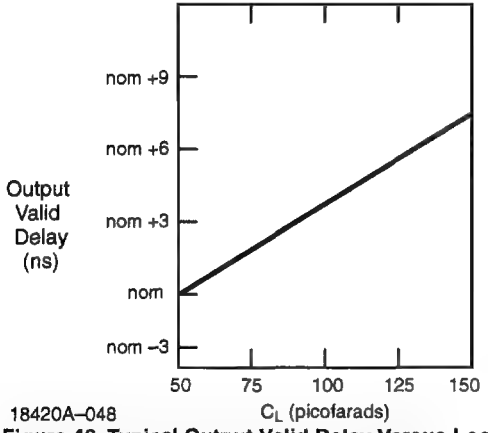


Figure 46. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L=50 pF)

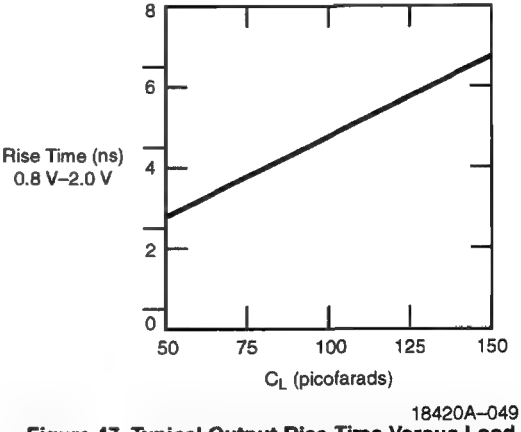


Figure 47. Typical Output Rise Time Versus Load Capacitance at Maximum Operating Temperature

DIFFERENCES BETWEEN THE Am386SE CPU AND THE Am386DE CPU

The following are the major differences between the Am386SE CPU and the Am386DE CPU:

1. The Am386SE CPU generates byte selects on $\overline{BHE}$ and $\overline{BLE}$ (like the 8086 and 80286) to distinguish the upper and lower bytes on its 16-bit data bus. The Am386DE CPU uses four byte selects, $\overline{BE3}$–$\overline{BE0}$, to distinguish between the different bytes on its 32-bit bus.
2. The Am386SE CPU has no bus sizing option. The Am386DE CPU can select between either a 32-bit bus or a 16-bit bus by use of the $\overline{BS16}$ input. The Am386SE CPU has a 16-bit bus size.
3. The $\overline{NA}$ pin operation in the Am386SE CPU is identical to that of the $\overline{NA}$ pin on the Am386DE CPU with one exception: the Am386DE CPU $\overline{NA}$ pin cannot be

activated on 16-bit bus cycles (where $\overline{BS16}$ is Low in the Am386DE CPU case), whereas $\overline{NA}$ can be activated on any Am386SE CPU bus cycle.

4. The contents of all Am386SE CPU registers at reset are identical to the contents of the Am386DE CPU registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, that is,

   in Am386DE CPU, after reset

   DH = 3 indicates Am386DE CPU

   in Am386SE CPU, after reset

   DH = 23H indicates Am386SE CPU

5. The Am386DE CPU uses A31 and M/$\overline{IO}$ as selects for the math coprocessor. The Am386SE CPU uses A23 and M/$\overline{IO}$ as selects.
6. The Am386DE CPU prefetch unit fetches code in four-byte units. The Am386SE CPU prefetch unit reads two bytes as one unit (like the 80286). In $\overline{BS16}$ mode, the Am386DE CPU takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the Am386DE CPU will fetch all four bytes before addressing the new request.
7. Both Am386DE CPU and Am386SE CPU have the same logical address space. The only difference is that the Am386DE CPU has a 32-bit physical address space and the Am386SE CPU has a 24-bit physical address space. The Am386SE CPU has a physical memory address space of up to 16 Mbytes instead of the 4 Gbytes available to the Am386DE CPU. Therefore, in Am386SE CPU systems, the operating system must be aware of this physical memory limit and should allocate memory for applications programs within this limit. If an Am386DE CPU system uses only the lower 16 Mbytes of physical address, then there will be no extra effort required to migrate Am386DE CPU software to the Am386SE CPU. In spite of this difference in physical address space, the Am386SE CPU and Am386DE CPU can run the same operating systems and applications within their respective physical memory constraints.
8. The Am386SE CPU has an input called $\overline{FLT}$ which three-states all bidirectional and output pins, including HLDA, when asserted. It is used with ON-Circuit Emulation (ONCE).

INSTRUCTION SET

This section describes the instruction set. The Instruction Set Clock Count Summary lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within instructions.

Am386SE Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in the Instruction Set Clock Count Summary, by the processor clock period (e.g., 40 ns for the 25-MHz Am386SE CPU). The actual clock count of an Am386SE CPU program will average 5% more than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. *n* = number of times repeated.
3. *m* = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses

— If instructions access a misaligned 16-bit operand or 32-bit operand on even address add:

  2 x clocks for read or write

  4 x clocks for read and write

— If instructions access a 32-bit operand on odd address add:

  4 x clocks for read or write

  8 x clocks for read and write

Wait States

Wait states add 1 clock per wait state to instruction execution for each data access.

Am386SE Microprocessor Instruction Set Clock Count Summary

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
GENERAL DATA TRANSFER					
MOV = Move					
Register to Register/Memory	1 0 0 0 1 0 0 w | mod reg r/m	2/2	2/2*	b	h
Register/Memory to Register	1 0 0 0 1 0 1 w | mod reg r/m	2/4	2/4*	b	h
Immediate to Register/Memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | immediate data	2/2	2/2*	b	h
Immediate to Register (short form)	1 0 1 1 w reg | immediate data	2	2		
Memory to Accumulator (short form)	1 0 1 0 0 0 0 w | full displacement	4*	4*	b	h
Accumulator to Memory (short form)	1 0 1 0 0 0 1 w | full displacement	2*	2*	b	h
Register/Memory to Segment Register	1 0 0 0 1 1 1 0 | mod sreg 3 r/m	2/5	22/23	b	h, i, j
Segment Register to Register/Memory	1 0 0 0 1 1 0 0 | mod sreg r/m	2/2	2/2	b	h
MOVSX = Move with Sign Extension					
Register from Register/Memory	0 0 0 0 1 1 1 1 | 1 0 1 1 1 1 1 w | mod reg r/m	3/6*	3/6*	b	h
MOVZX = Move with Zero Extension					
Register from Register/Memory	0 0 0 0 1 1 1 1 | 1 0 1 1 0 1 1 w | mod reg r/m	3/6*	3/6*	b	h
PUSH = Push					
Register/Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	5/7*	7/9*	b	h
Register (short form)	0 1 0 1 0 reg	2	4	b	h
Segment Register (ES,CS,SS, or DS) (short form)	0 0 0 sreg 2 1 1 0	2	4	b	h
Segment Register (ES, CS, SS, DS, FS, or GS)	0 0 0 0 1 1 1 1 | 1 0 sreg 3 0 0 0	2	4	b	h
Immediate	0 1 1 0 1 0 s 0 | immediate data	2	4	b	h
PUSHA = Push All	0 1 1 0 0 0 0 0	18	34	b	h
POP = Pop					
Register/Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	5/7	7/9	b	h
Register (short form)	0 1 0 1 1 reg	6	6	b	h
Segment Register (ES, CS, SS, or DS)	0 0 0 sreg 2 1 1 1	7	25	b	h, i, j
Segment Register (ES, CS, SS, DS, FS, or GS)	0 0 0 0 1 1 1 1 | 1 0 sreg 3 0 0 1	7	25	b	h, i, j
POPA = Pop All	0 1 1 0 0 0 0 1	24	40	b	h
XCHG = Exchange					
Register/Memory with Register	1 0 0 0 0 1 1 w | mod reg r/m	3/5**	3/5**	b, f	f, h
Register with Accumulator (short form)	1 0 0 1 0 reg	3	3		
IN = Input From:					
Fixed Port	1 1 1 0 0 1 0 w | port number	12*	6*/26*		s/t, m
Variable Port	1 1 1 0 1 1 0 w	13*	7*/27*		s/t, m

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
OUT = Output To:					
Fixed Port	1 1 1 0 0 1 1 w | port number	10*	4*/24*		s/t, m
Variable Port	1 1 1 0 1 1 1 w	11*	5*/25*		s/t, m
LEA = Load EA to Register	1 0 0 0 1 1 0 1 | mod reg r/m	2	2		
SEGMENT CONTROL					
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1 | mod reg r/m	7*	26*/28*	b	h, i, j
LES = Load Pointer to ES	1 1 0 0 0 1 0 0 | mod reg r/m	7*	26*/28*	b	h, i, j
LFS = Load Pointer to FS	0 0 0 0 1 1 1 1 | 1 0 1 1 0 1 0 0 | mod reg r/m	7*	26*/28*	b	h, i, j
LGS = Load Pointer to GS	0 0 0 0 1 1 1 1 | 1 0 1 1 0 1 0 1 | mod reg r/m	7*	26*/28*	b	h, i, j
LSS = Load Pointer to SS	0 0 0 0 1 1 1 1 | 1 0 1 1 0 0 1 0 | mod reg r/m	7*	26*/28*	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	1 1 1 1 1 0 0 0	2	2		
CLD = Clear Direction Flag	1 1 1 1 1 1 0 0	2	2		
CLI = Clear Interrupt Enable Flag	1 1 1 1 1 0 1 0	8	8		m
CLTS = Clear Task Switched Flag	0 0 0 0 1 1 1 1 | 0 0 0 0 0 1 1 0	5	5	c	l
CMC = Complement Carry Flag	1 1 1 1 0 1 0 1	2	2		
LAHF = Load AH into Flag	1 0 0 1 1 1 1 1	2	2		
POPF = Pop Flags	1 0 0 1 1 1 0 1	5	5	b	h, n
PUSHF = Push Flags	1 0 0 1 1 1 0 0	4	4	b	h
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0	3	3		
STC = Set Carry Flag	1 1 1 1 1 0 0 1	2	2		
STD = Set Direction Flag	1 1 1 1 1 1 0 1				
STI = Set Interrupt Enable Flag	1 1 1 1 1 0 1 1	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	0 0 0 0 0 0 d w | mod reg r/m	2	2		
Register to Memory	0 0 0 0 0 0 0 w | mod reg r/m	7**	7**	b	h
Memory to Register	0 0 0 0 0 0 1 w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0 0 0 0 0 1 0 w | immediate data	2	2		
ADC = Add with Carry					
Register to Register	0 0 0 1 0 0 d w | mod reg r/m	2	2		

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
ADC = Add with Carry (continued)					
Register to Memory	0 0 0 1 0 0 0 w | mod reg r/m	7**	7**	b	h
Memory to Register	0 0 0 1 0 0 1 w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0 0 0 1 0 1 0 w | immediate data	2	2		
INC = Increment					
Register/Memory	1 1 1 1 1 1 1 w | mod 0 0 0 r/m	2/6**	2/6**	b	h
Register (short form)	0 1 0 0 0 reg	2	2		
SUB = Subtract					
Register from Register	0 0 1 0 1 0 d w | mod reg r/m	2	2		
Register from Memory	0 0 1 0 1 0 0 w | mod reg r/m	7**	7**	b	h
Memory from Register	0 0 1 0 1 0 1 w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	1 0 0 0 0 0 s w | mod 1 0 1 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0 0 1 0 1 1 0 w | immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	0 0 0 1 1 0 d w | mod reg r/m	2	2		
Register from Memory	0 0 0 1 1 0 0 w | mod reg r/m	7**	7**	b	h
Memory from Register	0 0 0 1 1 0 1 w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0 0 0 1 1 1 0 w | immediate data	2	2		
DEC = Decrement					
Register/Memory	1 1 1 1 1 1 1 w | reg 0 0 1 r/m	2/6	2/6	b	h
Register (short form)	0 1 0 0 1 reg	2	2		
CMP = Compare					
Register with Register	0 0 1 1 1 0 d w | mod reg r/m	2	2		
Memory with Register	0 0 1 1 1 0 0 w | mod reg r/m	5*	5*	b	h
Register with Memory	0 0 1 1 1 0 1 w | mod reg r/m	6*	6*	b	h
Immediate with Register/Memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | immediate data	2/5*	2/5*	b	h
Immediate with Accumulator (short form)	0 0 1 1 1 1 0 w | immediate data	2	2		
NEG = Change Sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	2/6*	2/6*	b	h
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1	4	4		

* If CPL≤ IOPL ** If CPL> IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1	4	4		
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1	4	4		
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1	4	4		
MUL = Multiply (unsigned)					
Accumulator with Register Memory	1 1 1 1 0 1 1 w | mod 1 0 0 r/m				
Multiplier –Byte		12–17/15–20*	2–17/15–20*	b, d	d, h
–Word		12–25/15–28*	2–25/15–28*	b, d	d, h
–Doubleword		12–41/17–46*	2–41/17–46*	b, d	d, h
IMUL = Integer Multiply (signed)					
Accumulator with Register Memory	1 1 1 1 0 1 1 w | mod 1 0 1 r/m				
Multiplier –Byte		12–17/15–20*	2–17/15–20*	b, d	d, h
–Word		12–25/15–28*	2–25/15–28*	b, d	d, h
–Doubleword		12–41/17–46*	2–41/17–46*	b, d	d, h
Register with Register/Memory	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 1 1 | mod reg r/m				
Multiplier –Byte		12–17/15–20*	2–17/15–20*	b, d	d, h
–Word		12–25/15–28*	2–25/15–28*	b, d	d, h
–Doubleword		12–41/17–46*	2–41/17–46*	b, d	d, h
Register/Memory with Immediate to Register	0 1 1 0 1 0 s 1 | mod reg r/m | immediate data				
–Word		13–26	13–26/14–27	b, d	d, h
–Doubleword		13–42	13–42/16–45	b, d	d, h
DIV = Divide (unsigned)					
Accumulator by Register/Memory	1 1 1 1 0 1 1 w | mod 1 1 0 r/m				
Divisor –Byte		14/17	14/17	b, e	e, h
–Word		22/25	22/25	b, e	e, h
–Doubleword		38/43	38/43	b, e	e, h
IDIV = Integer Divide (signed)					
Accumulator by Register/Memory	1 1 1 1 0 1 1 w | mod 1 1 1 r/m				
Divisor –Byte		19/22	19/22	b, e	e, h
–Word		27/30	27/30	b, e	e, h
–Doubleword		43/46	43/46	b, e	e, h
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	19	19		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	17	17		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0	3	3		
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1	2	2		
LOGIC					
Shift/Rotate Instruction					
Not Through Carry **(ROL, ROR, SAL, SAR, SHL, and SHR)**					
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	3/7**	3/7**	b	h

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
LOGIC (continued)					
Not Through Carry (**ROL, ROR, SAL, SAR, SHL, and SHR**) –(continued)					
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	3/7*	3/7*	b	h
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m (1)	3/7*	3/7*	b	h
Through Carry (**RCL and RCR**)					
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m (1)	9/10*	9/10*	b	h
	TTT / **Instruction**: 000 ROL; 001 ROR; 010 RCL; 011 RCR; 100 SHL/SAL; 101 SHR; 111 SAR				
SHLD = Shift Left Double					
Register/Memory by Immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 0 | mod reg r/m (1)	3/7**	3/7**		
Register/Memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 1 | mod reg r/m	3/7**	3/7**		
SHRD = Shift Right Double					
Register/Memory by Immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 0 | mod reg r/m (1)	3/7**	3/7**		
Register/Memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 1 | mod reg r/m	3/7**	3/7**		
AND = And					
Register to Register	0 0 1 0 0 0 d w | mod reg r/m	2	2		
Register to Memory	0 0 1 0 0 0 0 w | mod reg r/m	7**	7**	b	h
Memory to Register	0 0 1 0 0 0 1 w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | immediate data	2/7*	2/7**	b	h
Immediate to Accumulator (short form)	0 0 1 0 0 1 0 w | immediate data	2	2		
TEST = And Function to Flags, No Result					
Register/Memory and Register	1 0 0 0 0 1 0 w | mod reg r/m	2/5*	2/5*	b	h
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | immediate data	2/5*	2/5*	b	h
Immediate Data and Accumulator (short form)	1 0 1 0 1 0 0 w | immediate data	2	2		
OR = Or					
Register to Register	0 0 0 0 1 0 d w | mod reg r/m	2	2		
Register to Memory	0 0 0 0 1 0 0 w | mod reg r/m	7**	7**	b	h
Memory to Register	0 0 0 0 1 0 1 w | mod reg r/m	6*	6*	b	h

* If CPL≤IOPL ** If CPL>IOPL

(1) Immediate 8-Bit Data

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
LOGIC (continued)					
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0 0 0 0 1 1 0 w | immediate data	2	2		
XOR = Exclusive Or					
Register to Register	0 0 1 1 0 0 d w | mod reg r/m	2	2		
Register to Memory	0 0 1 1 0 0 0 w | mod reg r/m	7**	7**	b	h
Memory to Register	0 0 1 1 0 0 1 w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0 0 1 1 0 1 0 w | immediate data	2	2		
NOT = Invert Register/Memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	2/6**	2/6**	b	h
STRING MANIPULATION					
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w	10*	10*	b	h
INS = Input Byte/Word from DX Port	0 1 1 0 1 1 0 w	15	9*/29**	b	s/t, h, m
LODS = Load Byte/Word to AL/AX/EAX	1 0 1 0 1 1 0 w	5	5*	b	h
MOVE = Move Byte/Word	1 0 1 0 0 1 0 w	7	7**	b	h
OUTS = Output Byte/Word to DX Port	0 1 1 0 1 1 1 w	14	8*/28**	b	s/t, h, m
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w	7*	7*	b	h
STOS = Store Byte/Word from AL/AX/EX	1 0 1 0 1 0 1 w	4*	4*	b	h
XLAT = Translate String	1 1 0 1 0 1 1 1	5*	5*		h
REPEATED STRING MANIPULATION					
Repeated by Count in CX or ECX					
REPE CMPS = Compare String (Find non-match)	1 1 1 1 0 0 1 1 | 1 0 1 0 0 1 1 w	5+9n**	5+9n**	b	h
REPNE CMPS = Compare String (Find match)	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 1 w	5+9n**	5+9n**	b	h
REP INS = Input String	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	13+6n*	7+6n*/ 27+6n**	6	s/t, h, m
REP LODS = Load String	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	5+6n*	5+6n*	b	h
REP MOVS = Move String	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	7+4n*	7+4n*	b	h
REP OUTS = Output String	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	12+5n*	6+5n*/ 26+5n**	b	s/t, h, m
REPE SCAS = Scan String (Find non-AL/AX/EAX)	1 1 1 1 0 0 1 1 | 1 0 1 0 1 1 1 w	5+8n*	5+8n*	b	h
REPNE SCAS = Scan String (Find AL/AX/EAX)	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 1 w	5+8n*	5+8n*	b	h
REP STOS = Store String	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	5+5n*	5+5n*	b	h

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format				Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
BIT MANIPULATION								
BSF = Scan Bit Forward	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	mod reg r/m	(1)	10 + 3n*	10 + 3n*	b	h
BSR = Scan Bit Reverse	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 1	mod reg r/m		10 + 3n*	10 + 3n*	b	h
BT = Test Bit								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 0 r/m	(1)	3/6*	3/6*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 0 0 0 1 1	mod reg r/m		3/12*	3/12*	b	h
BTC = Test Bit and Complement								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 1 1 r/m	(1)	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 1	mod reg r/m		6/13*	6/13*	b	h
BTR = Test Bit and Reset								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 1 0 r/m	(1)	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 1	mod reg r/m		6/13*	6/13*	b	h
BTS = Test Bit and Set								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 1 r/m	(1)	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 0 1 0 1 1	mod reg r/m		6/13*	6/13*	b	h
CONTROL TRANSFER								
CALL = Call								
Direct Within Segment	1 1 1 0 1 0 0 0	full displacement			7 + m*	9 + m*	b	r
Register/Memory Indirect Within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			7 + m* / 10 + m*	9 + m* / 12 + m*	b	h, r
Direct Intersegment	1 0 0 1 1 0 1 0	unsigned full offset, selector			17 + m*	42 + m*	b	j, k, r
Protected Mode Only (Direct Intersegment)								
Via Call Gate to Same Privilege Level						64 + m		h, j, k, r
Via Call Gate to Different Privilege Level (No Parameters)						98 + m		h, j, k, r
Via Call Gate to Different Privilege Level (x Parameters)						106+8x+m		h, j, k, r
From 80286 Task to 80286 TSS						285		h, j, k, r
From 80286 Task to Am386SE CPU TSS						310		h, j, k, r
From Am386SE CPU Task to 80286 TSS						285		h, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS						392		h, j, k, r
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m			30 + m	46 + m	b	h, j, k, r
Protected Mode Only (Indirect Intersegment)								
Via Call Gate to Same Privilege Level						68 + m		h, j, k, r
Via Call Gate to Different Privilege Level (No Parameters)						102 + m		h, j, k, r
Via Call Gate to Different Privilege Level (x Parameters)						110+8x+m		h, j, k, r
From 80286 Task to 80286 TSS								h, j, k, r
From 80286 Task to Am386SE CPU TSS								h, j, k, r
From Am386SE CPU Task to 80286 TSS								h, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS						399		h, j, k, r

* If CPL≤IOPL ** If CPL>IOPL

(1) Immediate 8-Bit Data

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
CONTROL TRANSFER (continued)					
JMP = Unconditional Jump					
Short	11101011 | 8-bit displacement	7 + m	7 + m		r
Direct within Segment	11010001 | full displacement	7 + m	7 + m		r
Register/Memory Indirect Within Segment	11111111 | mod 100 r/m	9 + m / 14 + m	9 + m / 14 + m	b	h, r
Direct Intersegment	11101010 | unsigned full offset, selector	16 + m	31 + m		j, k, r
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			53 + m		h, j, k, r
From 80286 Task to 80286 TSS					h, j, k, r
From 80286 Task to Am386SE CPU TSS					h, j, k, r
From Am386SE CPU Task to 80286 TSS					h, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS					h, j, k, r
Indirect Intersegment	11111111 | mod 101 r/m	17 + m	31 + m	b	h, j, k, r
Protected Mode Only (Indirect Intersegment)			49 + m		h, j, k, r
Via Call Gate to Same Privilege Level					h, j, k, r
From 80286 Task to 80286 TSS					h, j, k, r
From 80286 Task to Am386SE CPU TSS					h, j, k, r
From Am386SE CPU Task to 80286 TSS			328		h, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS					h, j, k, r
RET = Return from Call					
Within Segment	11000011		12 + m	b	g, h, r
Within Segment Adding Immediate to SP	11000010 | 16-bit displacement		12 + m	b	g, h, r
Intersegment	11001011		36 + m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010 | 16-bit displacement		36 + m	b	g, h, j, k, r
Protected Mode Only (RET): to Different Privilege Level					
Intersegment			72		h, j, k, r
Intersegment Adding Immediate to SP			72		h, j, k, r
CONDITIONAL JUMPS (Note: Times are Jump "Taken or Not Taken")					
JO = Jump on Overflow					
8-bit Displacement	01110000 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000000 | full displacement	7 + m or 3	7 + m or 3		r
JNO = Jump on Not Overflow					
8-bit Displacement	01110001 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000001 | full displacement	7 + m or 3	7 + m or 3		r

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
CONDITIONAL JUMPS (continued)					
JB/JNAE = Jump on Below/Not Above or Equal					
8-bit Displacement	01110010 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000010 | full displacement	7 + m or 3	7 + m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal					
8-bit Displacement	01110011 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000011 | full displacement	7 + m or 3	7 + m or 3		r
JE/JZ = Jump on Equal/Zero					
8-bit Displacement	01110100 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000100 | full displacement	7 + m or 3	7 + m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero					
8-bit Displacement	01110101 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000101 | full displacement	7 + m or 3	7 + m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above					
8-bit Displacement	01110110 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000110 | full displacement	7 + m or 3	7 + m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-bit Displacement	01110111 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000111 | full displacement	7 + m or 3	7 + m or 3		r
JS = Jump on Sign					
8-bit Displacement	01111000 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001000 | full displacement	7 + m or 3	7 + m or 3		r
JNS = Jump on Not Sign					
8-bit Displacement	01111001 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001001 | full displacement	7 + m or 3	7 + m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-bit Displacement	01111010 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001010 | full displacement	7 + m or 3	7 + m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-bit Displacement	01111011 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001011 | full displacement	7 + m or 3	7 + m or 3		r

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
CONDITIONAL JUMPS (continued)					
JL/JNGE = Jump on Less/Not Greater or Equal					
8-bit Displacement	01111100 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001100 | full displacement	7 + m or 3	7 + m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-bit Displacement	01111101 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001101 | full displacement	7 + m or 3	7 + m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-bit Displacement	01111110 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001110 | full displacement	7 + m or 3	7 + m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-bit Displacement	01111111 | 8-bit displacement	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001111 | full displacement	7 + m or 3	7 + m or 3		r
JCXZ = Jump on CX Zero*	11100011 | 8-bit displacement	9 + m or 5	9 + m or 5		r
JECXZ = Jump on ECX Zero	11100011 | 8-bit displacement	9 + m or 5	9 + m or 5		r
LOOP = Loop CX Times	11100010 | 8-bit displacement	11 + m	11 + m		r
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displacement	11 + m	11 + m		r
LOOPNZ/LOOPNE = Loop while Not Zero	11100000 | 8-bit displacement	11 + m	11 + m		r
CONDITIONAL BYTE SET (Note: Times Are Register/Memory)					
SETO = Set Byte on Overflow					
To Register/Memory	00001111 | 10010000 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	00001111 | 10010001 | mod 0 0 0 r/m	4/5*	4/5*		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	00001111 | 10010010 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNB = Set Byte on Not Below/Above or Equal					
To Register/Memory	00001111 | 10010011 | mod 0 0 0 r/m	4/5*	4/5*		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 | 10010100 | mod 0 0 0 r/m	4/5*	4/5*		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 | 10010101 | mod 0 0 0 r/m	4/5*	4/5*		h

* Address Size Prefix differentiates JCXZ from JECXZ

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
CONDITIONAL BYTE SET (continued)					
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 10010110 mod000 r/m	4/5*	4/5*		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 10010111 mod000 r/m	4/5*	4/5*		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 10011000 mod000 r/m	4/5*	4/5*		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 10011001 mod000 r/m	4/5*	4/5*		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 10011010 mod000 r/m	4/5*	4/5*		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 10011011 mod000 r/m	4/5*	4/5*		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 10011100 mod000 r/m	4/5*	4/5*		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 01111101 mod000 r/m	4/5*	4/5*		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 10011110 mod000 r/m	4/5*	4/5*		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 10011111 mod000 r/m	4/5*	4/5*		h
ENTER = Enter Procedure	11001000 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		14	14	b	h
L > 1		17 + 8 (n – 1)	17 + 8 (n – 1)	b	h
LEAVE = Leave Procedure	11001001	4	4	b	h
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set					
If OF = 1	11001110	35		b, e	
If OF = 0		3	3	b, e	

* If CPL≤IOPL ** If CPL>IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
INTERRUPT INSTRUCTIONS (continued)					
INT = Interrupt:					
Type Specified					
Type 3					
Bound = Interrupt 5 if Detected Value Out of Range	0 1 1 0 0 0 1 0 | mod reg r/m				
If Out of Range		44		b, e	e,g,h,j,k,r
If In Range		10	10	b, e	e,g,h,j,k,r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate to Same Privilege Level					
Via Interrupt or Trap Gate to Different Privilege Level			71		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			111		g, j, k, r
From 80286 Task to Am386SE CPU TSS via Task Gate			438		g, j, k, r
From Am386SE CPU Task to 80286 TSS via Task Gate			465		g, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS via Task Gate			440		g, j, k, r
			467		g, j, k, r
INT: Type 3					
Via Interrupt or Trap Gate to Same Privilege Level					
Via Interrupt or Trap Gate to Different Privilege Level			71		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			111		g, j, k, r
From 80286 Task to Am386SE CPU TSS via Task Gate			382		g, j, k, r
From Am386SE CPU Task to 80286 TSS via Task Gate			409		g, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS via Task Gate			384		g, j, k, r
			411		g, j, k, r
INTO					
Via Interrupt or Trap Gate to Same Privilege Level					
Via Interrupt or Trap Gate to Different Privilege Level			71		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			111		g, j, k, r
From 80286 Task to Am386SE CPU TSS via Task Gate			384		g, j, k, r
From Am386SE CPU Task to 80286 TSS via Task Gate			328		g, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS via Task Gate			Am386DE		g, j, k, r
			413		g, j, k, r
BOUND					
Via Interrupt or Trap Gate to Same Privilege Level					
Via Interrupt or Trap Gate to Different Privilege Level			71		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			111		g, j, k, r
From 80286 Task to Am386SE CPU TSS via Task Gate			358		g, j, k, r
			388		g, j, k, r

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
INTERRUPT INSTRUCTIONS (continued)					
BOUND (continued)					
From Am386SE CPU Task to 80286 TSS via Task Gate			368		g, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS via Task Gate			398		g, j, k, r
INTERRUPT RETURN					
IRET = Interrupt Return	1 1 0 0 1 1 1 1	24			g,h,j,k,r
Protected Mode Only (IRET)					
Via Interrupt or Trap Gate to Same Privilege Level (within Task)			42		g,h,j,k,r
Via Interrupt or Trap Gate to Different Privilege Level (within Task)			86		g,h,j,k,r
From 80286 Task to 80286 TSS			285		h, j, k, r
From 80286 Task to Am386SE CPU TSS			318		h, j, k, r
From Am386SE CPU Task to 80286 TSS			328		h, j, k, r
From Am386SE CPU Task to Am386SE CPU TSS			377		h, j, k, r
PROCESSOR CONTROL					
HLT = Halt	1 1 1 1 0 1 0 0	5	5		i
MOV = Move To and From Control/Debug/Test Registers					
CR0/CR2/CR3 from Register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 0 | 1 1 eee reg	10/4/5	10/4/5		i
Register from CR3–CR0	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 0 | 1 1 eee reg	5	6		i
DR3–DR0 from Register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 1 | 1 1 eee reg	22	22		i
DR7–DR6 from Register	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 1 1 | 1 1 eee reg	16	16		i
Register from DR7–DR6	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 1 | 1 1 eee reg	14	14		i
Register from DR3–DR0	0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 1 | 1 1 eee reg	22	22		i
NOP = No Operation	1 0 0 1 0 0 0 0	3	3		i
WAIT = Wait until BUSY pin is negated	1 0 0 1 1 0 1 1	6	6		i
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	1 1 0 1 1 T T T | mod L L L r/m TTT and LLL bits are op-code information for coprocessor.	See 387SX Data sheet for clock counts			h
PREFIX BYTES					
Address Size Prefix	0 1 1 0 0 1 1 1	0	0		

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
PREFIX BYTES (continued)					
LOCK = Bus Lock Prefix	11110000	0	0		m
Operand Size Prefix	01100110	0	0		
Segment Override Prefix					
CS	00101110	0	0		
DS	00111110	0	0		
ES	00100110	0	0		
FS	01100100	0	0		
GS	01100101	0	0		
SS	00110110	0	0		
PROTECTION CONTROL					
ARPL = Adjust Requested Privilege Level					
From Register/Memory	01100011 | mod reg r/m	N/A	20/21**	a	h
LAR = Load Access Rights					
From Register/Memory	00001111 | 00000010 | mod reg r/m	N/A	15/16*	a	g, h, j, p
LGDT = Load Global Descriptor					
Table Register	00001111 | 00000001 | mod 0 1 0 r/m	11*	11*	b, c	h, l
LIDT = Load Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 0 1 1 r/m	11*	11*	b, c	h, l
LLDT = Load Local Descriptor					
Table Register to Register/Memory	00001111 | 00000000 | mod 0 1 0 r/m	N/A	20/24*	a	g, h, j, l
LMSW = Load Machine Status Word					
From Register/Memory	00001111 | 00000001 | mod 1 1 0 r/m	10/13	10/13*	b, c	h, l
LSL = Load Segment Limit					
From Register/Memory	00001111 | 00000011 | mod reg r/m				
Byte-Granular Limit		N/A	20/21*	a	g, h, j, p
Page-Granular Limit		N/A	25/26*	a	g, h, j, p
LTR = Load Task Register					
From Register/Memory	00001111 | 00000000 | mod 0 0 1 r/m	N/A	23/27*	a	g, h, j, l
SGDT = Store Global Descriptor					
Table Register	00001111 | 00000001 | mod 0 0 0 r/m	9*	9*	b, c	h
SIDT = Store Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 0 0 1 r/m	9*	9*	b, c	h

* If CPL ≤ IOPL ** If CPL > IOPL

Am386SE Microprocessor Instruction Set Clock Count Summary (continued)

Instruction	Format	Clock Count: Real Address Mode	Clock Count: Protected Address Mode	Notes: Real Address Mode	Notes: Protected Address Mode
PROTECTION CONTROL (continued)					
SLDT = Store Local Descriptor Table Register					
To Register/Memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 0 0 r/m	N/A	2/2*	a	h
SMSW = Store Machine Status Word	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 1 0 0 r/m	2/2*	2/2*	b, c	h, l
STR = Store Task Register					
To Register/Memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 0 1 r/m	N/A	2/2*	a	h
VERR = Verify Read Access					
Register/Memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 1 0 0 r/m	N/A	10/11*	a	g, h, j, p
VERW = Verify Write Access	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 1 0 1 r/m	N/A	15/16*	a	g, h, j, p

* If CPL ≤ OPL ** If CPL > OPL

Instruction Notes for Instruction Set Summary

Notes a through c apply to Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in Exception 6 (invalid op-code).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS limit (FFFFH). Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to Real Address Mode and Protected Address Mode:

d. The Am386SE CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:
Actual Clock = if $m < > 0$, then max ([$\log_2 |m|$], 3) + b clocks;
= if $m = 0$, then 3 + b clocks

In this formula, m is the multiplier, and
b = 9 for register to register;
b = 12 for memory to register;
b = 10 for register with immediate to register;
b = 11 for memory with immediate to register.

e. An exception may occur, depending on the value of the operand.

f. $\overline{\text{LOCK}}$ is automatically asserted, regardless of the presence or absence of the LOCK prefix.

g. $\overline{\text{LOCK}}$ is asserted during descriptor table accesses.

Notes h through r apply to Protected Address Mode only:

h. Exception 13 fault will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or an access rights violation. If a stack limit is violated, an Exception 12 occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an Exception 13 fault. The segment's descriptor must indicate "present" or Exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an Exception 12 occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an Exception 13, if an applicable privilege rule is violated.

l. An Exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An Exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL field of the flag register is updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an Exception 13 fault will occur before the ESC instruction is executed. An Exception 12 fault will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an Exception 13 fault will occur.

s/t. The instruction will execute in s clocks if CPL ≤ IOPL. If CPL>IOPL, the instruction will take t clock.

Instruction Encoding

Overview

All instruction encodings are subsets of the general instruction format shown in the Am386SE Microprocessor Instruction Set Clock Count Summary Table (pages 72 thru 86). Instructions consist of one or two primary op-code bytes, possibly an address specifier consisting of the mod r/m byte and scaled index byte, a displacement if required, and an immediate data field if required.

Within the primary op-code(s), smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary op-code byte(s). This byte (mod r/m) specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte (scale-index-base byte) follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16, or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 48 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but Figure 48 does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the op-code bytes themselves. Table 19 is a complete list of all fields appearing in the Instruction Set. Further ahead, following Table 19, are detailed tables for each field.

32-Bit Extensions of the Instruction Set

With the Am386SE CPU, the 8086/80186/80286 Instruction Set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types; and, 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses, when executing that code segment. In the Real Address Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Am386SE CPU when operating in this mode (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any op-code bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed

Table 19. Fields Within Instructions

Field Name	Description	Number of Bits
w	Specifies if data is byte or full size (full size is either 16 or 32 bits)	1
d	Specifies direction of data operation	1
s	Specifies if an immediate data field must be sign-extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (effective address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS and ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, and GS	3
tttn	For Conditional Instructions, specifies a condition asserted or a condition negated	4

Note: *Table 19 shows encoding of individual instructions.*

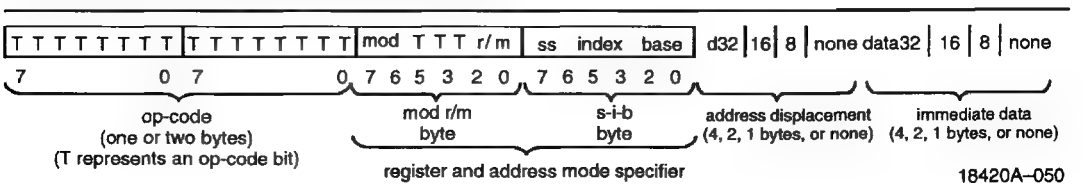


Figure 48. General Instruction Format

before the op-code bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value opposite from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all modes, including the Real Address Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high order of the extended registers.

Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary op-code bytes, or as the reg field of the mod r/m byte, or as the r/m field of the mod r/m byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
101	SI	ESI
101	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations		
	Function of w Field	
reg	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
	Function of w Field	
reg	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field, allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Am386SE CPU FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is predetermined, the addressing mode for the current instruction is specified by addressing bytes following the primary op-code. The primary addressing byte is the mod r/m byte, and a second byte of addressing information, the s-i-b (scale-index-base) byte, can be specified.

The s-i-b byte is specified when using 32-bit addressing mode, the mod r/m byte has r/m = 100, and mod = 00, 01, or 10. When the s-i-b byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the mod r/m byte, also contains three bits (shown as TTT in Figure 48) sometimes used as an extension of the primary op-code. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address, while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the mod r/m byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the mod r/m byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-Bit Address Mode with mod r/m Byte

mod r/m	Effective Address
00 000	DS: [BX + SI]
00 001	DS: [BX + DI]
00 010	SS: [BP + SI]
00 011	DS: [BP + DI]
00 100	DS: [SI]
00 101	DS: [DI]
00 110	DS: d16
00 111	DS: [BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + SI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	Register—See Below
11 001	Register—See Below
11 010	Register—See Below
11 011	Register—See Below
11 100	Register—See Below
11 101	Register—See Below
11 110	Register—See Below
11 111	Register—See Below

Register Specified by r/m During 16-Bit Data Operations		
	Function of w Field	
mod r/m	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
	Function of w Field	
mod r/m	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-Bit Address Mode with mod r/m Byte (no s-i-b byte present)

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	Register—See Below
11 001	Register—See Below
11 010	Register—See Below
11 011	Register—See Below
11 100	Register—See Below
11 101	Register—See Below
11 110	Register—See Below
11 111	Register—See Below

Register Specified by reg or r/m During 32-Bit Data Operations		
	Function of w Field	
mod r/m	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Register Specified by reg or r/m During 16-Bit Data Operations		
	Function of w Field	
mod r/m	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Encoding of 32-Bit Address Mode (mod r/m byte and s-i-b byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

Note:
Mod field in mod r/m byte; ss, index, and base fields in s-i-b byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

Index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg (see note)
101	EBP
110	ESI
111	EDI

Note:
When index field is 100, indicating no index register, then ss field must equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

Encoding of Operation Direction (d) Field

In many two-operand instructions, the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory ⇐ Register reg Field indicates Source Operand; mod r/m or mod ss index base indicates Destination Operand.
1	Register ⇐ Register/Memory reg Field indicates Destination Operand; mod r/m or mod ss index base indicates Source Operand.

Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extended Data8 to Fill 16-Bit or 32-Bit Destination	None

Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0), or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	No Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Not Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug, and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

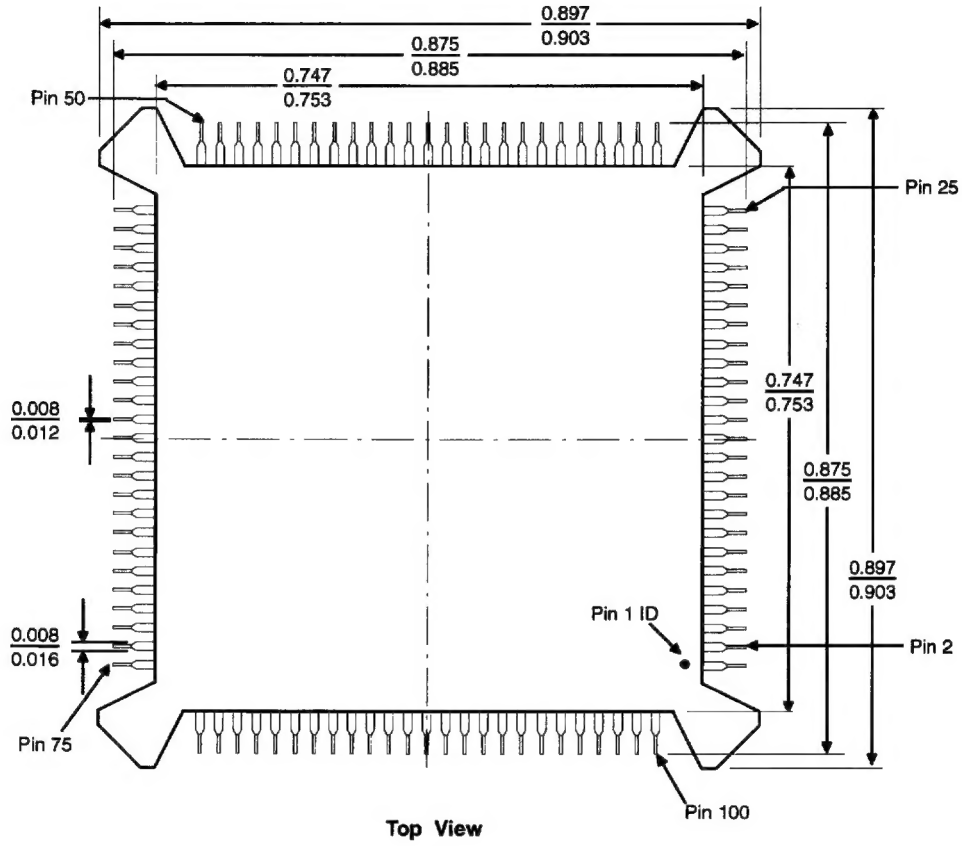
When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	

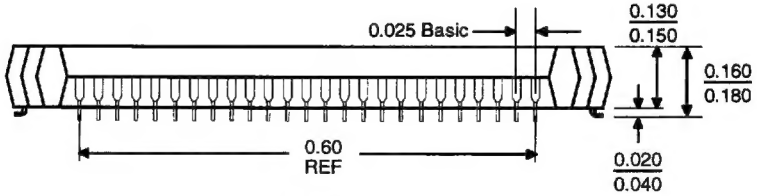
PHYSICAL DIMENSIONS

For reference only. All dimensions measured in inches unless otherwise noted. BSC is an ANSI standard for Basic Space Centering.

PQB 100—Plastic Quad Flat Pack; Trimmed and Formed



Top View



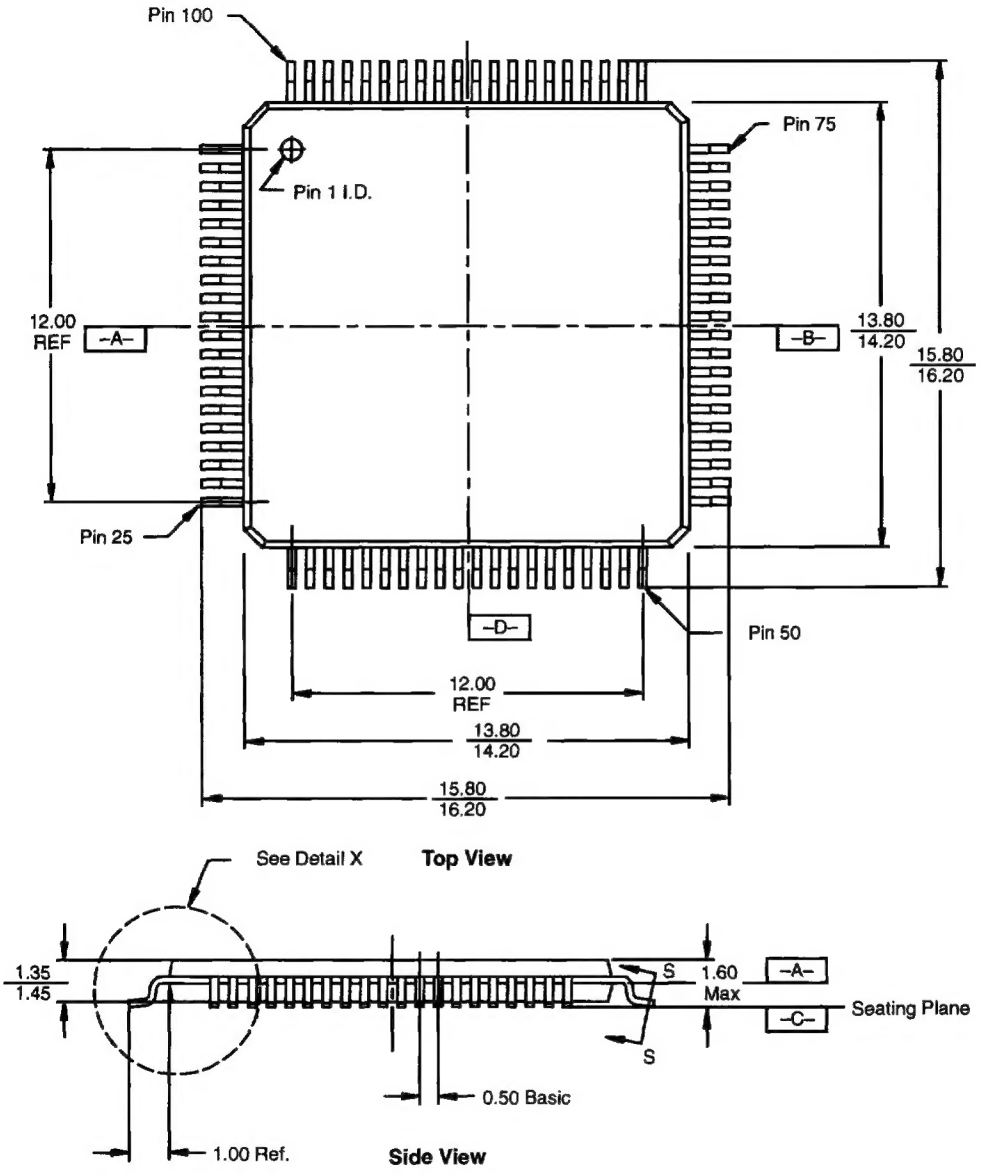
Side View

20010A
CL85
08/04/93 MH

Notes:

1. *All measurements are in inches unless otherwise noted.*
2. *Not to scale. For reference only.*

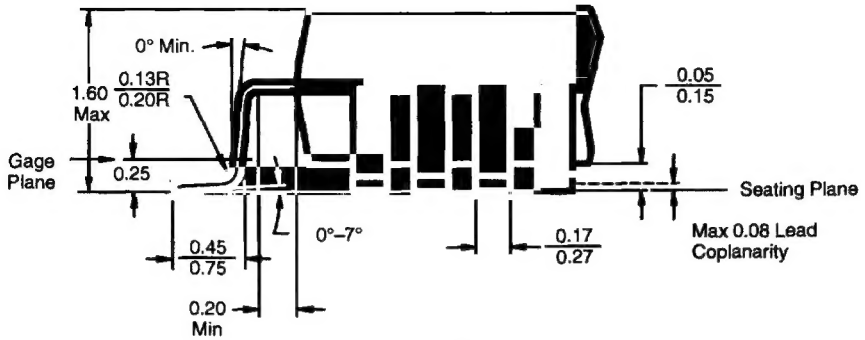
PQT 100—Metric Thin Quad Flat Pack—Plastic Package; Trimmed and Formed



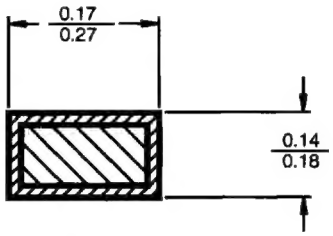
Notes:

1. *All measurements are in millimeters unless otherwise noted.*
2. *Not to scale. For reference only.*

PQT 100 (continued)



Detail X



Section S–S

Notes:
1. Not to scale. For reference only.

North American

ALABAMA	(205) 882-9122
ARIZONA	(602) 242-4400
CALIFORNIA,	
Culver City	(310) 645-1524
Newport Beach	(714) 752-6262
Sacramento(Roseville)	(916) 786-6700
San Diego	(619) 560-7030
San Jose	(408) 922-0500
Woodland Hills	(818) 878-9988
CANADA, Ontario,	
Kanata	(613) 592-0060
Willowdale	(416) 222-7800
COLORADO	(303) 741-2900
CONNECTICUT	(203) 264-7800
FLORIDA,	
Clearwater	(813) 530-9971
Boca Raton	(407) 361-0050
Orlando (Longwood)	(407) 862-9292
GEORGIA	(404) 449-7920
IDAHO	(208) 377-0393
ILLINOIS,	
Chicago (Itasca)	(708) 773-4422
Naperville	(708) 505-9517
MARYLAND	(301) 381-3790
MASSACHUSETTS	(617) 273-3970
MINNESOTA	(612) 938-0001
NEW JERSEY,	
Cherry Hill	(609) 662-2900
Parsippany	(201) 299-0002
NEW YORK,	
Brewster	(914) 279-8323
Rochester	(716) 425-8050
NORTH CAROLINA	
Charlotte	(704) 875-3091
Raleigh	(919) 878-8111
OHIO,	
Columbus (Westerville)	(614) 891-6455
Dayton	(513) 439-0268
OREGON	(503) 245-0080
PENNSYLVANIA	(215) 398-8006
TEXAS,	
Austin	(512) 346-7830
Dallas	(214) 934-9099
Houston	(713) 376-8084

International

BELGIUM, Antwerpen	TEL	(03) 248 43 00
	FAX	(03) 248 46 42
FRANCE, Paris	TEL	(1) 49-75-10-10
	FAX	(1) 49-75-10-13
GERMANY,		
Bad Homburg	TEL	(06172)-24061
	FAX	(06172)-23195
München	TEL	(089) 45053-0
	FAX	(089) 406490
HONG KONG,	TEL	(852) 865-4525
Wanchai	FAX	(852) 865-4335
ITALY, Milano	TEL	(02) 3390541
	FAX	(02) 38103458
JAPAN,		
Tokyo	TEL	(03) 3346-7550
	FAX	(03) 3342-5196
Osaka	TEL	(06) 243-3250
	FAX	(06) 243-3253
KOREA, Seoul	TEL	(82) 2-784-0030
	FAX	(82) 2-784-8014

International (Continued)

LATIN AMERICA,		
Ft. Lauderdale	TEL	(305) 484-8600
	FAX	(305) 485-9736
SINGAPORE	TEL	(65) 3481188
	FAX	(65) 3480161
SWEDEN,		
Stockholm area	TEL	(08) 98 61 80
(Bromma)	FAX	(08) 98 09 06
TAIWAN, Taipei	TEL	(886) 2-7153536
	FAX	(886) 2-7122183
UNITED KINGDOM,		
Manchester area	TEL	(0925) 830380
(Warrington)	FAX	(0925) 830204
London area	TEL	(0483) 740440
(Woking)	FAX	(0483) 756196

North American Representatives

CANADA	
Burnaby, B.C. – DAVETEK MARKETING	(604) 430-3680
Kanata, Ontario – VITEL ELECTRONICS	(613) 592-0060
Mississauga, Ontario – VITEL ELECTRONICS	(905) 564-9720
Lachine, Quebec – VITEL ELECTRONICS	(514) 636-5951
ILLINOIS	
Skokie – INDUSTRIAL REPRESENTATIVES,INC	(708) 967-8430
IOWA	
LORENZ SALES	(319) 377-4666
KANSAS	
Merriam – LORENZ SALES	(913) 469-1312
Wichita – LORENZ SALES	(316) 721-0500
MEXICO	
Chula Vista – SONIKA ELECTRONICA	(619) 498-8340
Guadalajara – SONIKA ELECTRONICA	(523) 647-4250
Mexico City – SONIKA ELECTRONICA	(523) 754-6480
Monterrey – SONIKA ELECTRONICA	(523) 358-9280
MICHIGAN	
Holland – COM-TEK SALES, INC	(616) 335-8418
Brighton – COM-TEK SALES, INC	(313) 227-0007
MINNESOTA	
Mel Foster Tech. Sales, Inc.	(612) 941-9790
MISSOURI	
LORENZ SALES	(314) 997-4558
NEBRASKA	
LORENZ SALES	(402) 475-4660
NEW MEXICO	
THORSON DESERT STATES	(505) 883-4343
NEW YORK	
East Syracuse – NYCOM, INC	(315) 437-8343
Hauppauge – COMPONENT CONSULTANTS, INC	(516) 273-5050
OHIO	
Centerville – DOLFUSS ROOT & CO	(513) 433-6776
Westlake – DOLFUSS ROOT & CO	(216) 899-9370
PENNSYLVANIA	
RUSSELL F. CLARK CO.,INC.	(412) 242-9500
PUERTO RICO	
COMP REP ASSOC, INC	(809) 746-6550
UTAH	
FRONT RANGE MARKETING	(801) 288-2500
WASHINGTON	
ELECTRA TECHNICAL SALES	(206) 821-7442
WISCONSIN	
Brookfield – INDUSTRIAL REPRESENTATIVES,INC	(414) 574-9393

Advanced Micro Devices, Inc. One AMD Place, P.O. Box 3453, Sunnyvale, CA 94088-3453, USA
Tel: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE & LITERATURE ORDERING • TOLL FREE: (800) 222-9323 • (408) 749-5703

18420B 4/7/94
Con-8M-6/94-0 Printed in USA